MOVABLE BARRIER OPERATOR HAVING FORCE AND POSITION LEARNING CAPABILITY

BACKGROUND OF THE INVENTION

The invention relates in general to a movable barrier operator for opening and closing a movable barrier or door. More particularly, the invention relates to a garage door operator that can learn force and travel limits when installed and can simulate the temperature of its electric motor to avoid motor failure during operation.

A number of garage door operators have been sold over the years. Most garage door operators include a head unit containing a motor having a transmission connected to it, which may be a chain drive or a screw drive, which is coupled to a garage door for opening and closing the garage door. Such garage door openers also have included optical detection systems located near the bottom of the travel of the door to prevent the door from closing on objects or on persons that may be in the path of the door. Such garage door operators typically include a wall control which is connected via one or more wires to the head unit to send signals to the head unit to cause the head unit to open and close the garage door, to light a worklight or the like. Such prior art garage door operators also include a receiver and head unit for receiving radio frequency transmissions from a hand-held code transmitter or from a keypad transmitter which may be affixed to the outside of the garage or other structure. These ga age door operators typically include adjustable limit switches which cause the garage door to operate or to halt the motor when the travel of the door causes the limit switch to change state which may either be in the up position or in the down position. This prevents damage to the door as well damage to the structure supporting the door. It may be appreciated, however, that with different size garages and different size doors, the limits of travel must be custom set once the unit is placed within the garage. In the past, such units have had mechanically adjustable limit switches which

are typically set by an installer. The installer must go back and forth between the door, the wall switch and the head unit in order to make the adjustment. This, of course, is time consuming and results in the installer being forced to spend more time than is desirable to install the garage door operator.

A number of requirements are in existence from Underwriter's Laboratories, the Consumer Product Safety Commission and the like which require that garage door operators sold in the United States must, when in a closing mode and contacting an obstruction having a height of more than one inch, reverse and open the door in order to prevent damage to property and injury to persons. Prior art garage door operators also included systems whereby the force which the electric motor applied to the garage door through the transmission might be adjusted. Typically, this force is adjusted by a licensed repair technician or installer who obtained access to the inside of the head unit and adjusts a pair of potentiometers, one of which sets the maximal force to be applied during the closing portion of door operation, the other of which establishes the maximum force to be applied during the opening of door operation.

Such a garage door operator is exemplified by an operator taught in U.S. Patent No. 4,638,443 to Schindler. However, such door operators are relatively inconvenient to install and invite misuse because the homeowner, using such a garage door operator, if the garage door operator begins to bind or jam in the tracks, may likely obtain access to the head unit and increase the force limit. Increasing the maximal force may allow the door to move passed a binding point, but apply the maximal force at the bottom of its travel when it is almost closed where, of course, it should not.

Another problem associated with prior art garage door operators is that they typically use electric motors having thermostats connected in series with portions of

their windings. The thermostats are adapted to open when the temperature of the winding exceeds a preselected limit. The problem with such units is that when the thermostats open, the door then stops in whatever position it is then in and can neither be opened or closed until the motor cools, thereby preventing a person from exiting a garage or entering the garage if they need to.

SUMMARY OF THE INVENTION

The present invention is directed to a movable barrier operator which includes a head unit having an electric motor positioned therein, the motor being adapted to drive a transmission connectable to the motor, which transmission is connectable to a movable barrier such as a garage door. A wired switch is connectable to the head unit for commanding the head unit to open and close the door and for commanding a controller within the head unit to enter a learn mode. The controller includes a micro-controller having a non-volatile memory associated with it which can store force set points as well as digital end of travel positions within it. When the controller is placed in learn mode by appropriate switch closure from the wall switch, the door is caused to cycle open and closed. The force set point stored in the non-volatile memory is a relatively low set point and if the door is placed in learn mode and the door reaches a binding position, the set point will be changed by increasing the set point to enable the door to travel through the binding area. Thus, the set points will be dynamically adjusted as the door is in the learn, but the set points will not be changeable once the door is taken out of the learn mode, thereby preventing the force set point from being inadvertently increased, which might lead to property damage or injury. Likewise, the end of travel positions can be adjusted automatically when in the learn mode because if the door is halted by the controller, when the controller senses that the door

position has reached the previously set end of travel position, the door will then be commanded by a button push from the wall switch to keep travelling in the same direction, thereby incrementing or changing. The end of travel limits are set by pushing the learn button on the wall switch which causes the door to travel upward and continue travelling upward until the door has travelled as far as the operator wishes it to travel. The disables the learn switch by lifting his hand from the button. The up limit is then stored and the door is then moved toward the closed position. A pass point or position normalizing system consisting of a ring-like light interrupter attached to the garage door crosses the light path of an optical obstacle detector signalling instantaneously the position of the door and the door continues until it closes, whereupon force sensing in the door causes an auto-reverse to take place and then raises the door to the up position, the learn mode having been completed and the door travel limits having been set.

The movable barrier operator also includes a combination of a temperature sensor and microcontroller. The temperature sensor senses the ambient temperature within the head unit because it is positioned in proximity with the electric motor. When the electric motor is operated, a count is incremented in the microcontroller which is multiplied by a constant which is indicative of the speed at which the motor is moving. This incremented multiplied count is then indicative of the rise in temperature which the motor has experienced by being operated. The count has subtracted from it the difference between the simulated temperature and the ambient temperature and the amount of time which the motor has been switched off. The totality of which is multiplied by a constant. The remaining count then is an indication of the extant temperature of the motor. In the event that the temperature, as determined by the microcontroller, is relatively high, the unit provides a predictive function in that if an attempt

is made to open or close the garage door, prior to the door moving, the microcontroller will make a determination as to whether the single cycling of the door will add additional temperature to the motor causing it to exceed a set point temperature and, if so, will inhibit operation of the door to prevent the motor from being energized so as to exceed its safe temperature limit.

The movable barrier operator also includes light emitting diodes for providing an output indication to a user of when a problem may have been encountered with the door operator. In the event that further operation of the door operator will cause the motor to exceed its set point temperature, an LED will be illuminated as a result of the microcontroller temperature prediction indicating to the user that the motor is not operating because further operation will cause the motor to exceed its safe temperature limits.

It is a principal aspect of the present invention to provide a movable barrier operator which is able to quickly and automatically select end of travel positions.

It is another aspect of the present invention to provide a movable barrier operator which, upon installation, is able to quickly establish up and down force set points.

It is still another aspect of the present invention to provide a movable barrier operator which can determine the temperature of the motor based upon motor history and the ambient temperature of the head unit.

Other aspects and advantages of the invention will become obvious to one of ordinary skill in the art upon a perusal of the following specification and claims in light of the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a perspective view of a garage having mounted within it a garage door operator embodying the present invention;

FIG. 2 is a block diagram of a controller mounted within the head unit of the garage door operator employed in the garage door operator shown in FIG. 1;

FIG. 3 is a schematic diagram of the controller shown in block format in FIG. 2;

FIG. 4 is a schematic diagram of a receiver module shown in the schematic diagram of FIG. 3;

FIG. 5A-B are a flow chart of a main routine that executes in a microcontroller of the control unit;

FIGS. 6A-G are a flow diagram of a learn routine executed by the microcontroller;

FIGS. 7A-B are flow diagrams of a timer routine executed by the microcontroller;

FIGS. 8A-B are flow diagrams of a state routine representative of the current and recent state of the electric motor;

FIGS. 9A-B are a flow chart of a tachometer input routine and also determines the position of the door on the basis of the pass point system and input from the optical obstacle detector;

FIGS. 10A-C are flow charts of the switch input routines from the switch module; and

FIG. 11 is a schematic diagram of the switch module and the switch biasing circuit.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring now to the drawings and especially to FIG. 1, more specifically a movable barrier door operator or garage door operator is generally shown therein and referred to by numeral 10 includes a head unit 12 mounted within a garage 14. More specifically, the head unit 12 is mounted to the ceiling of the garage 14 and includes a rail 18 extending therefrom with a releasable trolley 20 attached having an arm 22 extending to a multiple paneled garage door 24 positioned for movement along a pair of door rails 26 and 28. The system includes a hand-held transmitter unit 30 adapted to send signals to an antenna 32 positioned on the head unit 12 and coupled to a receiver as will appear hereinafter. An external control pad 34 is positioned on the outside of the garage having a plurality of buttons thereon and communicate via radio frequency transmission with the antenna 32 of the head unit 12. A switch module 39 is mounted on a wall of the garage. The switch module 39 is connected to the head unit by a pair of wires 39a. The switch module 39 includes a learn switch 39b, a light switch 39c., a lock switch 39d and a command switch 39e. An optical emitter 42 is connected via a power and signal line 44 to the head unit. An optical detector 46 is connected via a wire 48 to the head unit 12. A pass point detector 49 comprising a bracket 49a and a plate structure 49b extending from the bracket has a substantially circular aperture 49c formed in the bracket, which aperture might also be square or rectangular. The pass point detector is arranged so that it interrupts the light beam on a bottom leg 49d and allows the light beam to pass through the aperture 49c. The light beam is again interrupted by the leg 49e, thereby signalling the controller via the optical detector 46 that the pass point detector attached to the door has moved passed a certain position allowing the controller to normalize or zero its position, as will be appreciated in more detail hereinafter.

As shown in FIG. 2, the garage door operator 10, which includes the head unit 12 has a controller 70 which includes the antenna 32. The controller 70 includes a power supply 72 which receives alternating current from an alternating current source, such as 110 volt AC, and converts the alternating current to +5 volts zero and 24 volts DC. The 5 volt supply is fed along a line 74 to a number of other elements in the controller 70. The 24 volt supply is fed along the line 76 to other elements of the controller 70. The controller 70 includes a super-regenerative receiver 80 coupled via a line 82 to supply demodulated digital signals to a microcontroller 84. The receiver is energized by a line 86 coupled to the line 74. The microcontroller is also coupled by a bus 86 to a non-volatile memory 88, which non-volatile memory stores set points and other customized digital data related to the operation of the control unit. An obstacle detector 90, which comprises the emitter 42 and infrared detector 46 is coupled via an obstacle detector bus 92 to the micro-controller. The obstacle detector bus 92 includes lines 44 and 48. The wall switch 39 is connected via the connecting wires 39a to a switch biasing module 96 which is powered from the 5 volt supply line 74 and supplies signals to and is controlled by the microcontroller via a bus 100 coupled to the microcontroller. The microcontroller, in response to switch closures, will send signals over a relay logic line 102 to a relay logic module 104 connected to an alternating current motor 106 having a power take-off shaft 108 coupled to the transmission 18 of the garage door operator. A tachometer 110 is coupled to the shaft 108 and provides a tachometer signal on a tachometer line 112 to the microcontroller 84. The tachometer signal being indicative of the speed of rotation of the motor.

The power supply 72 includes a transformer 130 which receives alternating current on leads 132 and 134 from an external source of alternating current. The transformer steps down the voltage to 24 volts and feeds

24 volts to a pair of capacitors 138 and 140 which provide a filtering function. A 24 volt filtered DC potential is supplied on the line 76 to the relay logic 104. The potential is fed through a resistor 142 across a pair of filter capacitors 144 and 146, which are connected to a 5 volt voltage regulator 150, which supplies regulated 5 volt output voltage across a capacitor 152 and a Zener diode 154 to the line 74.

Signals may be received by the controller at the antenna 32 and fed to the receiver 80. The receiver 80 includes a pair of inductors 170 and 172 and a pair of capacitors 174 and 176 that provide impedance matching between the antenna 32 and other portions of the receiver. An NPN transistor 178 is connected in common base configuration as a buffer amplifier. Bias to the buffer amplifier transistor 178 is provided by resistors 180. A resistor 188, a capacitor 190, a capacitor 192 and a capacitor 194 provide filtering to isolate a later receiver stage from the buffer amplifier 178. An inductor 196 also provides power supply buffering. The buffered RF output signal is supplied on a line 200, coupled between the collector of the transistor 178 and a receiver module 202 which is shown in FIG. 4. The lead 204 feeds into the unit 202 and is coupled to a biasing resistor 220. The buffered radio frequency signal is fed via a coupling capacitor 222 to a tuned circuit 224 comprising a variable inductor 226 connected in parallel with a capacitor 228. Signals from the tuned circuit 220 are fed on a line 230 to a coupling capacitor 232 which is connected to an NPN transistor 234 at its based 236. The transistor has a collector 240 and emitter 242. The collector 240 is connected to a feedback capacitor 246 and a feedback resistor 248. The emitter is also coupled to the feedback capacitor 246 and to a capacitor 250. The line 210 is coupled to a choke inductor 256 which provides ground potential to a pair of resistors 258 and 260 as well as a capacitor 262. The resistor 258 is connected to the base 236 of the transistor 234. The

resistor 260 is connected via an inductor 264 to the emitter 242 of the transistor. The output signal from the transistor is fed outward on a line 212 to an electrolytic capacitor 270.

As shown in FIG. 3, the capacitor 270 capacitively couples the demodulated radio frequency signal to a bandpass amplifier 280 to an average detector 282 which feeds a comparator 284. The comparator 284 also receives a signal directly from the bandpass amplifier 280 and provides a demodulated digital output signal on the line 82 coupled to the P32 pin of the Z86E21/61 microcontroller. The microcontroller is energized by the power supply 72 and also controlled by the wall switch 39 coupled to the microcontroller by the leads 100.

From time to time, the microcontroller will supply current to the switch biasing module 96.

The microcontroller operates under the control of a main routine as shown in FIGS. 5A and 5B. When the unit is powered up, a power on reset is performed in a step 300, the memory is cleared and a check sum from read-only memory within the microcontroller 84 is tested. In a step 302, if the check sum and the memory prove to be correct, control is transferred to a step 304, if not, control is transferred back to the step 300. In the step 304, the last non-volatile state, which is indicative of the state of the operator, that is whether the operator indicated the door was at its up limit, down limit or in the middle of its travel, is tested for in a step 304 and if the last state is a down limit, control is transferred to a step 306. If it was an up limit, control is transferred to a step 308. If it was neither a down nor an up limit, control is transferred to a step 310. In the step 306, the position is set as the down limit value and a window flag is set. The operation state is set as down limit. In a step 308, the position is set as up, the window flag is set and the operation state is set as up limit. In the step 310, the position is set as outside the normal range, 6 inches below

the secondary up limit. The operation state is set as stopped. Control is transferred from any of steps 306, 308 and 310 to a step 312 where a stored simulated motor temperature is read from the non-volatile memory 88. The temperature of a printed circuit board positioned within the head unit is read from the temperature sensor 120 which is supplied over a line 120a to the microcontroller. In order to read the PC board temperature, a pin P20 of the microprocessor is driven high, causing a high potential to appear on a line 120b which supplies a current through the RTD sensor 120 to a comparator 120c. A capacitor 120d connected to the comparator and to the temperature sensor, is grounded and charges up. The other input terminal to the comparator has a voltage divider 120e connected to it to supply a reference voltage of about 2.5 volts. Thus, the microcontroller starts a timer running when it brings line 120b high and interrogates a line 120f to determine its state. The line 120f will be driven high when the temperature at the junction of the RTD 120 and the capacitor 120d exceeds 2.5 volts. Thus, the time that it takes to charge the capacitor through the resistance is indicative of the temperature within the head unit and, in this manner, the PC board temperature is read and if the temperature as read is greater than the temperature retrieved from the non-volatile memory, the temperature read from the PC board is then stored as the motor temperature.

In a step 314, constants related to the receipt and processing of the demodulated signal on the line 82 are initialized. In a step 316, a test is made to determine whether the learn switch 39b had been activated within the last 30 seconds. If it has not, control is transferred back to the step 314.

In a step 318, a test is made to determine whether the command switch debounce timer has expired. If it has, control is transferred to a step 320. If it is not, control is transferred back to the step 314. In the

step 320, the learn limit cycle is begun as will be discussed in more detail as to FIGS. 6A through 6G. The main routine effectively has a number of interrupt routines coupled to it. In the event that a falling edge is detected on the line 112 from the tachometer, an interrupt routine related to the tachometer is serviced in the step 322. A timer interrupt occurs every 0.5 millisecond in a step 324 as shown in FIGS. 7A through 7B.

The obstacle detector 90 generates a pulse every 10 milliseconds during the time when the beam from the infrared emitter 42 has not been interrupted either by the pass point system 49 or by an obstacle, in a step 326 following which the obstacle detector timer is cleared in a step 328.

As shown in FIGS. 10A through 10C, operation of the switch biasing module 96 is controlled over the lines 100 by the microcontroller 84. The microcontroller 84, in the step 340, tests to determine whether an RS232 digital communications mode has been set. If it has, control is transferred to a step 342, as shown in FIG. 10C, testing whether data is stored in an output buffer to be output from the microcontroller. If it is, control is transferred to a step 344 outputting the next bit, which may include a start bit, from the output buffer and control is then transferred back to the main routine. In the event that there is no data in the data buffer, control is transferred to the step 346, testing whether data is being received over lines 100. If it is being received, control is transferred to a step 348 to receive the next bit into the input buffer and the routine is then exited. If not, control is transferred to a step 350. In the step 350, a test is made to determine whether a start bit for RS232 signalling has been received. If it has not, control is transferred to a return step 352. If it has, control is transferred to a step 354 in which a flag is set indicating that the start bit has been received and the routine is exited. As shown in FIG. 10A, if the response to the

decision block 340 is no, control is transferred to a decision step 360. The switch status counter is incremented and then a test is determined as to whether the contents of the counter are 29. If the switch counter is 29, control is transferred to a step 362 causing the counter to be zeroed. If the counter is not 29, control is transferred to a step 364, testing for whether the switch status is equal to zero. If the switch status is equal to zero, control is transferred to a step 366. In a step 366, a current source transistor 368, shown in FIG. 8, is switched on, drawing current through resistors 370 and 372 and feeding current out through a line 39a connected thereto to the switch module 39a and, more specifically, to a resistor 380, a 0.10 microfarad capacitor 382, a 1 microfarad capacitor 384, a 10 microfarad capacitor 386 and a switch terminal 388. The switch 39e is coupled to the switch terminal 388. The switch 39d may be selectively coupled to the capacitor 386. The switch 39b may be selectively coupled to the capacitor 384. The switch 39c may be selectively coupled to the capacitor 382. A light emitting diode 392 is connected to the resistor 380. Current flows through the resistor 380 and the light emitting diode 392 back to another one of the lines 39a and through a field effect transistor 398 to ground. In step 402, the sense input on a line 100 coupled to the transistor 398 is tested to determine whether the input is high. If the input is high immediately, that is indicative of the fact that switches 39b through 39e are all open and in a step 404, debounce timers are decremented for all switches and a got switch flag is set and the routine is exited in the event that the test of step 402 is negative. Control is then transferred to a step 406 testing after 10 milliseconds if the sense in output on the line 100 connected to the field effect transistor 398 is high, which would be indicative of the switch 39c having been closed. If it is high, the worklight timer is incremented, all other switch timers are decremented, the got switch flag is

set and the routine is exited. In the event that the decision in step 406 is in the negative, control is transferred to a step 410 and the routine is exited. In the event that the decision from step 364 is in the negative, control is transferred to a step 412 wherein the switch status is tested as to whether it is equal to one. If it is, control is transferred to a step 414 testing whether the sensed input on the line 100 connected to the field effect transistor is high. If it is, control is transferred to step 416 to set the got switch flag, after which in a step 418, the learn switch debouncer is incremented, all other switch counters are decremented, the got switch flag is set and the routine is exited. In the event that the answer to step 414 is in the negative, control is transferred to a return step 420.

In the event that the answer to step 412 is in the negative, control is transferred to a step 422, as shown in FIG. 10B. A test is made as to whether the switch status is equal to 10. If it is, control is transferred to a step 424 where the sense out input is tested as high.

Thus, the charging rate for the capacitors which, in effect, is sensed on the line 100 connected to the field effect transistor 398 which is coupled to ground, is indicative of which of the switches is closed because the switch 39c has a capacitor that charges at 10 times the rate of the capacitor 384 connected to 39b and 100 times the rate of the capacitor 386 selectively couplable to switch 39d.

After the switch measurement has been made, the transistor 368 is switched non-conducting by the line 368b and the field effect transistor 398 is switched non-conducting by a line 450 connected to its gate. A transistor 462, coupled via a resistor 464 to a line 466, is switched on, biasing a transistor 468 on, causing current to flow through a diagnostic light emitting diode 470 to a field effect transistor 472 which is switched on via a voltage on a line 474. In addition, the capacitors

386, 384 and 382, which may have been charged are discharged through the field effect transistor 472.

In order to perform all of the switching functions after the step 424 has been executed, control is transferred to a step 510 testing whether the got switch flag has been cleared. If it has, control is transferred to a step 512 in which the command timer is incremented and all other timers are decremented and the got switch flag is set and the routine is exited. If the got switch flag is cleared as indicated in the step 510, the routine is exited in the step 514. In the event that the sense input is measured as being high in the step 424, control is transferred to a step 516 where the vacation or lock flag counter is incremented and all other counters are decremented. The got switch flag is set and the routine is exited. In the event that the switch status equal 10 test in the step 422 is indicated to be no, control is then transferred to a step 520 testing whether the switch status is 11. If the switch status is 11, indicating that the routine has been swept through 11 times, control is transferred to a step 522 in which the field effect transistors 398 and 472 are both switched on, providing ground pads on both sides of the capacitors causing the capacitors to discharge and the routine is then exited. In the event that the step 520 test is negative, control is transferred to a step 524 testing whether the routine has been executed 15 times. If it has, control is transferred to a step 526 indicating that the bit which controls the status the light emitting diode 470, the diagnostic light emitting diode, has been set. If it has not been set, control is transferred to a step 528 wherein both transistors 368 and 468 are switched on and both the field effect transistors 398 and 472 are switched off. In order to test for short circuits between the source and drain electrodes of the field effect transistors 398 and 472 which might cause false operation signals to be supplied on the lines 100 to the microcontroller 84, resulting in

inadvertent operation of the electric motor. The routine is then exited. In the event that the test in step 526 indicates that the diagnostic LED bit has been set, control is transferred to a step 530. In the step 530, the transistors 468 and 472 are switched on allowing current to flow through the diagnostic LED 470. In the event that the test in step 524 is negative, a test is made in a step 532 as to whether the routine has been executed 26 times. If it has not, the routine is exited in a step 534. If it has, both of the field effect transistors 398 and 372 are switched on to connect all of the capacitors to ground to discharge the capacitors and the routine is exited.

As shown in FIGS. 7A and 7B, when the timer interrupt occurs as in step 324, control is transferred to a step 550 shown in FIG. 7A wherein a test is made to determine whether a 2 millisecond timer has expired. If it has not, control is transferred to a step 552 determining whether a 500 millisecond timer has expired. If the 500 millisecond timer has expired, control is transferred to a step 554 testing whether power has been switched on through the relay logic 104 to the electric motor 106. If the motor has been switched on, control is transferred to a step 556 testing whether the motor is stalled, as indicated by the motor power having been switched on and by the fact that pulses are not coming through on the line 112 from the tachometer 110. In the event that the motor has stalled, control is transferred to a step 558. In the step 558 the existing motor temperature indication, as stored in one of the registers of the microcontroller 84, has added to it a constant which is related to a motor characteristic which is added in when the motor is indicated to be stalled. In the event that the response to the step 556 is in the negative, indicating that the motor is not stalled, control is transferred to a step 560 wherein the motor temperature is updated by adding a running motor constant to the motor temperature. In the event that the response to the test in step 554 is in the negative, indicating that

motor power is not on and that heat is leaking out of the motor so that the temperature will be dropping, the new motor temperature is assigned as being equal to the old motor temperature, less the quantity of the old motor temperature, minus the ambient temperature measured from the RTD probe 120, the whole difference multiplied by a thermal decay fraction which is a number.

All of steps 558, 560 and 562 exit to a step 564 which test as to whether a 15 minute timer has timed out. If the timer has timed out, control is transferred to a step 566 causing the current, or updated motor temperature, to be stored in a non-volatile memory 88. If the 15 minute timer has not been timed out, control is transferred to a step 510, as shown in FIG. 7B. Step 566 also exits to step 568. A test is made in the step 568 to determine whether a obstacle detector interrupt has come in via step 326 causing the obstacle detector timer to have been cleared. If it has not, the period will be greater than 12 milliseconds, indicating that the obstacle detector beam has been blocked. If the obstacle detector beam, in fact, has been blocked, control is transferred to a step 570 to set the obstacle detector flag.

In the event that the response to step 568 is in the negative, the obstacle detector flag is cleared in the step 572 and control is transferred to a step 574. All operational timers, including radio timers and the like are incremented and the routine is exited.

In the event that the 2 millisecond timer tested for in the step 550 has expired, control is transferred to a step 576 which calls a motor operation routine. Following execution of the motor operation routine, control is transferred to the step 552. When the motor operation routine is called, as shown in FIG. 8A, a test is made in a step 580 to determine the status of the motor operation state variable which may indicate that the up limit has been reached. If the up limit or the down limit have been reached, the motor is causing the door to travel up or

down, the door has stopped in mid-travel or an auto-reverse delay indicating that the motor has stopped in mid-travel and will be switching into up travel shortly. In the event that there is an auto-reverse delay, control is transferred to a step 582, when a test is made for a command from one of the radio transmitters or from the wall control unit and, if so, the state of the motor is set indicating that the motor has stopped in mid-travel. Control is then transferred to a step 584 in which 0.50 second timer is tested to determine whether it has expired. If it has, the state is set to the up travel state following which the routine is exited in the step 586. In the event that the operation state is in the up travel state, as tested for in step 580, control is transferred to a step 588 testing for a command from a radio or wall control and if the command is received, the motor operational state is changed to stop in mid-travel. Control is transferred to a step 590. If the force period indicated is longer than that stored in an up array location, indicated by the position of the motor. The state of the door is indicated as stopped in mid-travel. Control is then transferred to a step 592 testing whether the current position of the door is at the up limit, then the state of the door is set as being at the up limit and control is transferred to a step 594 causing the routine to be exited, as shown in FIG. 8B.

In the event that the operational state tested for in the step 580 is indicated to be at the up limit, control is transferred to a step 596 which tests for a command from the radio or wall control unit and a test is made to determine whether the motor temperature is below a set point for the down travel motor temperature threshold. The state is set as being a down travel state. If the temperature value exceeds the threshold or set point temperature value, an output diagnostic flag is set for providing an output indication in another routine. Control is then transferred to a step 598, causing the routine to be exited. In the event that the down travel limit has

been reached, control is transferred to a step 600 testing for whether a command has come in from the radio or wall control and, if it has, the state is set as auto-reverse and the auto-reverse timer is cleared. Control is then transferred to a step 602 testing whether the force period, as indicated, is longer than the force period stored in the down travel array for the current position of the door. Auto-reverse is then entered at step 582 on a later iteration of the routine. Control is transferred to a step 604 to test whether the position of the door is at the down limit position and the pass point detector has already indicated that the door has swept the passed the pass point, the state is set as a down limit state and control is transferred to a step 606 testing for whether the door position is at the down limit position and testing for whether the pass point has been detected. If the pass point has not been detected, the motor operational state is set to auto-reverse, causing auto-reverse to be entered in a later routine and control is transferred to a step 608, exiting the main routine.

In the event that the block 580 indicates that the door is at the down limit, control is transferred to a step 610, testing for a command from the radio or wall control and testing the current motor temperature. If the current motor temperature is below the up travel motor temperature threshold, then the motor state variable is set as equal to up travel. If the temperature is above the threshold or set point temperature, a diagnostic code flag is then set for later diagnostic output and control is transferred to a return step 612. In the event that the motor operational state is indicated as being stopped in mid-travel, control is transferred to a step 614 which tests for a radio or wall control command and tests the motor temperature value to determine whether it is above or below a down travel motor temperature threshold. If the motor temperature is above the travel threshold, then the

door is left stopped in mid-travel and the routine is returned from in step 616.

In the event that the learn switch has been activated as tested for in step 316 and the command switch is being held down as indicated by the positive result from the step 318, the learn limit cycle is entered in step 320 and transfers control to a step 630, as shown in FIG. 6A, in step 630, the maximum force is set to a minimum value from which it can later be incremented, if necessary. The motor up and motor down controllers in the relay logic 104 are disabled. The relay logic 104 includes an NPN transistor 700 coupled to line 76 to receive 24 to 28 volts therefrom via a coil 702 of a relay 704 having relay contacts 706. A transistor 710 coupled to the micro-controller is also coupled to line 76 via a relay coil 714 and together comprise an up relay 718 which is connected via a lead 720 to the electric motor 106. A down transistor 730 is coupled via a coil 732 to the power supply 76. The down relay 732 has an armature 734 associated with it and is connected to the motor to drive it down. Respective diodes 740 and 742 are connected across coils 714 and 732 to provide protection when the transistors 710 and 730 are switched off. In the step 632, both the transistors 710 and 730 are switched off, interrupting either up motor power or down motor power to the electric motor 106 and the microcontroller delays for 0.50 second. Control is then transferred to a step 634, causing the relay 704 to be switched on, delivering power to an electric light or worklight 750 associated with the head unit. The up motor relay 716 is switched on. A 1 second timer is also started which inhibits testing of force limits due to the inertia of the door as it begins moving. Control is then transferred to a step 636, testing for whether the 1 second timer has timed out and testing for whether the force period is longer than the force limit setting. If both conditions have occurred, control is transferred to a step 640 as shown in FIG. 6B. If either

the 1 second timer has not timed out or the force period is not longer than the force limit setting, control is transferred to a step 638 which tests whether the command switch is still being held down. If it is, control is transferred back to step 636. If it is not, control is transferred to the step 640. In step 640, both the up transistor 710 and the down transistor 730 are causing both the up motor and down motor command from the relay logic to be interrupted and a delay of 0.50 second is taken and the position counter is cleared. Control is then transferred to a step 640 in which the transistor 730 is commanded to switch on, starting the motor moving down and the 1 second force ignore timer is started running. A test is made in a step 642 to determine whether the command switch has been activated again. If it has, the force limit setting is increased in a step 644 following which control is then transferred back to the step 632. If the command switch is not being held down, control is then transferred to a step 646, testing whether the 1 second force ignore timer has timed out. The last 32 rpm pulses indicative of the force are ignored and a force period from the previous pulse is accepted as the down force. Control is then transferred to a step 648 and a test is made to determine whether the movable barrier is at the pass point as indicated by the pass point detector 49 interacting with the optical detector 46. Control is then transferred to a step 650. The position counter is complemented and the complemented value is stored as the up limit following which the position counter is cleared and a pass point flag is set. Control is then transferred back to the step 642. In the event that the result of the test in step 648 is negative, control is transferred to a step 652 which tests whether the 1 second force delay timer has expired and whether the force period is greater than the force limit setting, indicating that the force has exceeded. If both of those conditions have occurred, control is transferred to a step 654 which tests whether the pass point flag has been set.

If it has not been set, control is transferred to a step 656, wherein the position counter is complemented and the complemented value is saved as the up limit and the position counter is cleared. In the event that the pass point flag has been set, control is transferred to a step 658. In the event that the test in step 652 has been negative, control is transferred to a step 660 which tests the value of the obstacle reverse flag. If the obstacle reverse flag has not been set, control is transferred to the step 642 shown on FIG. 6B. If the flag has been set, control is transferred to the step 654.

In a step 658, both transistors 710 and 730 are switched off interrupting up and down power from the relays to the electric motor 106 and halting the motor and the microcontroller then delays for 0.50 second. Control is then transferred to a step 660. In step 660, the transistor 710 is switched on switching on the up relay causing the motor to be turned to drive the door upward and the 1 second force ignore timer is started. Control is transferred to a decision step 662 testing for whether the command switch is set. If the command switch is set, control is transferred back to the step 664 causing the force limit setting to be increased, following which control is transferred to the step 632, interrupting the motor outputs. If the command switch has not been set, control is transferred to the step 664 causing the maximum force from the 33rd previous reading to be saved as the up force, following which control is transferred to a decision block 666 which tests for whether the 1 second force ignore timer has expired and whether the force period is longer than the force limit setting. If both conditions are true, control is transferred to a step 668. If not, control is transferred to a step 670 which tests for whether the door position is at the up limit. If the door position is at the up limit, control is transferred to the step 668, switching off both of the motor outputs to halt the door and delaying for 0.50 second. If the position tested in

step 670 is not at the upper limit, control is transferred back to the step 662. Following step 668, control is transferred to the step 676 during which the command switch is tested. If the command switch is set, control is transferred back to the step 644 causing the force limit setting to be increased and ultimately to the step 632 which switches off the motor outputs and delays for 0.50 second. If the command switch has not been set, control is transferred to a step 678. If the position counter indicates that the door is presently at a point where a force transition normally occurs or where force settings are to change, and the 1 second force ignore timer has expired, the 33rd previous maximum force is stored and the down force array is filled with the last 33 force measurements. Control is then transferred to a step 680 which tests for whether the obstacle detector reverse flag has been set. If it has not been set, control is transferred to a step 682 which tests for whether the 1 second force ignore timer has expired and whether the force period is longer than the force limit setting. If both those conditions are true, control is transferred to a step 684 which tests for the pass point being set. If the pass point flag was not set, control is transferred to the step 688. In the event that the obstacle reverse flag is set, control is also transferred to the step 688. In the event that the decision block 682 is answered in the negative, control is transferred back to the step 676. If the pass point flag has been set as tested for in the step 684, control is transferred to the step 686 wherein the current door position is saved as the down limit position. In step 688, both the motor output transistors 710 and 730 are switched off, interrupting up and down power to the motor and a delay occurs for 0.50 second. Control is then transferred to the step 690 wherein the up transistor 710 is switched on, causing the up relay to be actuated, providing up power to the motor and the 1 second force ignore timer begins running. In the step 692, a test is

made for whether the command has been set again. If it has, control is transferred back to the step 644, as shown in FIG. 6B, and following that to the step 632, as shown in FIG. 6A. If the command switch has not been set, control is transferred to the step 694 which tests for whether the position counter indicates that the door is at a sectional force transition point or barrier and the 1 second force ignore timer has expired. If both those conditions are true, the maximum force from the last sectional barrier is then loaded. Control is then transferred to a decision step 696 testing for whether the 1 second force ignore timer has timed out and whether the force period is indicated to be longer than the force period limit setting. If both of those conditions are true, control is then transferred to a step 698 causing the motor output transistors 710 and 730 to be switched off and all data is stored in the non-volatile memory 88 and the routine is exited. In the event that decision is indicated to be in the negative from the decision step 696, control is transferred to a step 697 which tests whether the door position is presently at the up limit position. If it is, control is then transferred to the step 698. If it is not, control is transferred to the step 692.

In the event that the rpm interrupt step 322, as shown in FIG. 5B, is executed, control is then transferred to a step 800, as shown in FIG. 9A. In step 800, the time duration from the last rpm pulse from the tachometer 110 is measured and saved as a force period indication. Control is then transferred to a decision block. Control is transferred to the step 802, in which the operator state variable is tested. In the event that the operator state variable indicates that the operator is causing the door to travel down, the door is at the down limit or the door is in the auto-reverse mode, control is transferred to a step 804 causing the door position counter to be incremented. In the event that the door operator state indicates that the door is travelling upward, has reached its up limit or

has stopped in mid-travel, control is transferred to a step 806 which causes the position counter to be decremented. Control is then transferred to a decision step 808 in which the pass point pattern testing flag is tested for whether it is set. If it is set, control is transferred to a step 810 which tests a timer to determine whether the maximum pattern time allotted by the system has expired. In the event that the pass point pattern testing flag is not set, control is transferred to a step 812, testing for whether the optical obstacle detector flag has been set. If is not, the routine is exited in a step 814. If the obstacle detector flag has been set, control is transferred to a step 816 wherein the pattern testing flag is set and the routine is exited. In the event that the maximum pattern time has timed out. As tested for in the step 810, control is transferred to a step 820 wherein the optical reverse flag is set and the routine is exited. In the maximum pattern time has not expired, a test is made in a step 822 for whether the microcontroller has sensed from the obstacle detector that the beam has been blocked open within a correct timing sequence indicative of the pass point detection system. If it has not, the routine is exited in a step 824. If it has, control is transferred to a step 826. Testing for whether a window flag has been set. As to whether the rough position of the door would indicate that the pass point should have been encountered. f the window flag has been set, control is transferred to a step 828, testing for whether the position is within the window flag position. If it has, control is transferred to a step 832, causing the position counter to be cleared or renormalized or zeroed, setting the window flag and set a flag indicating that the pass point has been found, following which the routine is exited. In the event that the position is now within the window as tested for in step 828, the obstacle reverse flag is set in a step 830 and the routine is exited. In the event that the test made in step

326 indicates that the window flag has not been set, control is then transferred directly to the step 832.

While there has been illustrated and described a particular embodiment of the present invention, it will be appreciated that numerous changes and modifications will occur to those skilled in the art, and it is intended in the appended claims to cover all those changes and modifications which fall within the true spirit and scope of the present invention.

```
;    11 =          Switch state to discharge P3 = 0101 XXXX FOR NEW LAYOUT
;
;  Clear the radio codes from RTO
;       or new code flag "output RTO"
;
;       Note temp is temp +40
;
;
;       change temp adder for running reset    change stall temp adder
;
;       Note remove from set any clr switch_data and clr radio_cmd
;
;       add fill before the 101 org
;               dn_limit and 2X up_limit commented out
;
;       REMOVED THE UP LIMIT & DOWN LIMIT
;       CONDITIONAL OF RPM CAUSING FORCED UP STATE
;
;
;
;      45  46           4A  4B             4F
;       .......          ........         .
;       .     .          .      .        .
;  44  .    47 .     49 .     4C .    4E .
;     .        .       . .      .      .
;42*43*        ----  ----        ----  ----
;             ----  ----         ----  ----
;               ......            ......
;                 48                4D
;
;      Jog
;
;     30    31        .......
;       .....        .
;           .       .
;       32 ......... 33
;
; Position is done from rpm on   direction is assumed from the state of the system
;
;       State           Assumed Direction
;
;       Autoreverse     Down
;       Up_Direction    Up
;       Up_Position     Up
;       Reset           Up
;       Dn_Direction    Down
;       Dn_Position     Down
;       Stop            Up
;
;The position counter is zeroed at the end of the patterned IR interruption
; in the down direction and increases
; from there to the max position which is the down limit
; the patterned position is from the bottom of the door having a 3/4 inch bar,
; a 3/4 inch space then another 3/4 inch bar
```

```
; since the gdo is giving 30 pulses for ever rotation of the upper sprokel we have
; 6 touth => 20 rpm pulses
; 8 touth => 15 rpm pulses
;10 touth => 12 rpm pulses
;
;The set up will be done from the program mode being set and the wall control being activated
; the door will travel up first then down and reverses off a .5 inch obstruction
; at the reversal point the position number is the max position
; Startup shall be in the up direction
;
;
;
; RS 232 is done from the wall control baud is 1200
;
;
;
; Secondary state machine for learning
; 42 Stop All Travel
; 43 Delay .5 seconds
; 44 Set up direction
; 45 At up limit
; 46 Delay .5 second
; 47 Down travel
; 48 Arev
; 49 Up travel
; 4A At up limit
; 4B Delay .5 seconds
; 4C Down travel
; 4D Arev
; 4E Up travel
; 4F At up limit
; else clear
;---------------------------------------------------------------------------
;          NON-VOL MEMORY MAP
;---------------------------------------------------------------------------
;
;          00          A0
;          01          A0
;          02          A1
;          03          A1
;          04          A2
;          05          A2
;          06          A3
;          07          A3
;          08          A4
;          09          A4
;          0A          A5
;          0B          A5
;          0C          A6
;          0D          A6
;          0E          A7
;          0F          A7
;          10          A8
;          11          A8
;          12          A9
```

```
;         13        A9
;         14        A10
;         15        A10
;         16        A11
;         17        A11
;         18        B
;         19        B
;         1A        C
;         1B        C.
;         1C        CYCLE COUNTER 1ST 16 BITS
;         1D        CYCLE COUNTER 2ND 16 BITS
;         1E        VACATION FLAG
;
;                   Vacation Flag . Last Operation
;                   0000            XXXX  in vacation
;                   1111            XXXX out of vacation
;
;         1F        A MEMORY ADDRESS LAST WRITTEN
;
; Max speed 1800 RPM => 150 pulses / sec * 27 seconds => 4050 pulses max => 15 groups
;
;         20        Up Force 1      0000-EFFF
;         21        Up Force 2      FFFF-FF00
;         22        Up Force 3      FEFF-FE00
;         23        Up Force 4      FDFF-FD00
;         24        Up Force 5      FCFF-FC00
;         25        Up Force 6      FBFF-FB00
;         26        Up Force 7      FAFF-FA00
;         27        Up Force 8      F9FF-F900
;         28        Up Force 9      F8FF-F800
;         29        Up Force 10     F7FF-F700
;         2A        Up Force 11     F6FF-F600
;         2B        Up Force 12     F5FF-F500
;         2C        Up Force 13     F4FF-F400
;         2D        Up Force 14     F3FF-F300
;         2E        Temperature of motor
;         2F        Up Limit setting
;
;         30        Down Force 1    0000-EFFF
;         31        Down Force 2    FFFF-FF00
;         32        Down Force 3    FEFF-FE00
;         33        Down Force 4    FDFF-FD00
;         34        Down Force 5    FCFF-FC00
;         35        Down Force 6    FBFF-FB00
;         36        Down Force 7    FAFF-FA00
;         37        Down Force 8    F9FF-F900
;         38        Down Force 9    F8FF-F800
;         39        Down Force 10   F7FF-F700
;         3A        Down Force 11   F6FF-F600
;         3B        Down Force 12   F5FF-F500
;         3C        Down Force 13   F4FF-F400
;         3D        Down Force 14   F3FF-F300
;         3E        Last operation and reason
;         3F        Down Limit setting
;
;---------------------------------------------------------------------
```

RS232 DATA

INPUT	OUTPUT	
"0" 30H	Switches and mode	
	0011XXX0	Command switch not closed
	0011XXX1	Command switch closed
	0011XX0X	Light switch not closed
	0011XX1X	Light switch closed
	0011X0XX	Vacation switch not closed
	0011X1XX	Vacation switch closed
"1" 31H	System status	
	0011XXX0	Not in vacation mode
	0011XXX1	In vacation mode
	0011XX0X	Worklight off
	0011XX1X	Worklight on
	0011X0XX	No Aobs Errors
	0011X1XX	Aobs errors
"2" 32H	RPM period	
"3" 33H		
	0011XXX0	Learn switch not closed
	0011XXX1	Learn switch closed
	0011XX0X	Not in learn mode
	0011XX1X	In learn mode
	0011X0XX	Window not active
	0011X1XX	Window active
"4" 34H	Radio memory codes Page 00 32 BYTES	
"5" 35H	Radio memory codes Page 10 32 BYTES	
"6" 36H	Up force table. Up limit, and motor temp.	
"7" 37H	Down force table, down limit, and last operation	
"8" 38H	MEMORY TEST AND ERASE ALL!! 00 OK FF ERROR	
"9" 39H	Set program mode	
"A" 41H	Present position of travel Position = First byte * 256 + second byte	
"B" 42H	Down limit position Down limit = First byte * 256 + second byte	

"C" 43H	Up limit position Up limit = First byte * 256 + second byte
"D" 44H	Max force Max force = First byte * 256 + second byte
"E" 45H	Force setting up direction Force = First byte * 256 + second byte
"F" 46H	Force setting down direction Force = First byte * 256 + second byte
"G" 47H	Window size
"H" 48H	Window active "0" off "1" on
"I" 49H	Give a command sets the command debouncer for normal command send a "P" then "I" for learning limit send "Q9I" then a "P" when at up position
"J" 4AH	READ the temperature of the logic board +40C
"K" 4BH	READ the temperature of the motor +40C
"L" 4CH	9 For normal operation not in learn 0 Min force 1 2 3 Max forces
"M" 4DH	Vacation switch command
"N" 4EH	Light switch command
"O" 4FH	Force adder
"P" 50H	Clear the command debouncer
"Q" 51H	Set the command debouncer
"R" 52H	Last Radio code received if new else nothing
"S" 53H	Temperature PCB ASCII
"T" 54H	Temp motor ASCII Temperature PCB ASCII
"U"55H	Wake up code to set rs232 mode Returns the version
"V" 56H	State ASCII "0" Autorevers delay "1" Traveling in the up direction "2" At the up position

"3" Error
"4" Traveling in the down direction
"5" At the down position
"6" Stopped in mid travel

"W" 57H	Reason ASCII
	"0" Command
	"1" Radio command
	"2" Force
	"3" Protector
	"4" Autoreverse delay
	"5" Limits
	"6" Early limits
	"7" Timeout
	"8" RPM forcing up
	"9" Cmd held to limits
	"A" B code to the limits
	"B" Over temperature
	"F" No Pass Point
"X" 58H	Fault code ASCII
"Y" 59H	Straps ASCII
	00110X00 10 tooth
	00110X01 9.5 tooth
	00110X10 6 tooth
	00110X11 8 tooth
	001100XX Thermal protector off
	001101XX Thermal protector on
"Z" 5AH	Fixed table window off

Rs232 learn limits
output "Q91" when at up limit position "P"

DIAG

---

1) AOBS shorted
2) AOBS open / miss aligned
3) Protector intermittenent
4) Over temp
5) Memory bad
6) No RPM in the first second
7) RPM forced a reverse

DOG 2

```
;
;       DOG 2 IS A SECONDARY WATCHDOG USED TO
;       RESET THE SYSTEM IF THE LOWEST LEVEL "MAINLOOP"
;       IS NOT REACHED WITHIN A 3 SECOND
;
;
;
;------------------------------------------------------------------
;       Conditions
;------------------------------------------------------------------
;
;
Yes                     .equ    1h
No                      .equ    0h
E21                     .equ    Yes         ; E21 or C33 8K
DownToLimits            .equ    No          ; command held bypass
TempMeasureFlag         .equ    Yes         ; else set temperature to 85C
ForceTempCompFlag       .equ    Yes         ; else set force to .5mS adder
ThermalProtectorFlag    .equ    Yes         ; else skip test for motor temperature
P5BlockFlag             .equ    No          ; need .5 inch block
AOBSBypass              .equ    No          ; Protector not bypassed from cmd of B
PassProtector           .equ    Yes         ; is the pass point the protector or
                                            ; the switch pass point
RTD                     .equ    Yes         ; is the thermal device a RTD
;
;
;
;------------------------------------------------------------------
;       EQUATE STATEMENTS
;------------------------------------------------------------------


MINAR                   .equ    7D          ; min # rpm pulse for interruption
MAXAR                   .equ    150d        ; max # rpm pulse for pass point
UpDownTime              .equ    03d         ;

;------------------------------------------------------------------
; distance verses tooth
; Pulses / Inch = Pulses / Motor rev * Motor rev / Shaft rev * Shaft rev / Teeth * Teeth / Inch
; for 6 tooth   = 5 * 16 * 1/6 * 2 = 26..666
; for 8 teeth   = 5 * 16 * 1/8 * 2 = 20
; for 9.5 tooth = 5 * 16 * 1/9.5 * 2 = 16.84
; for 8 teeth   = 5 * 16 * 1/10 * 2 = 16
;------------------------------------------------------------------

L10Hi                   .equ    00h         ; 10 tooth
L10Lo                   .equ    8D

L9P5Hi                  .equ    00H         ; 9.5 tooth
L9P5Lo                  .equ    9D

L8Hi                    .equ    00h         ; 8 tooth
L8Lo                    .equ    10D

L6Hi                    .equ    00h         ; 6 tooth
L6Lo                    .equ    13D
```

```
TempRunIncHi        .equ    00h
TempRunIncLo        .equ    5Ch         ; rate of temperature increase running
                                        ; every second
TempStallIncHi      .equ    00h         ;
TempStallIncLo      .equ    0B8h        ; rate of temperature increase stalled
                                        ; every second
T27Adder            .equ    005H        ; adder if running when reset

UpSetMaxTemp        .equ    160d        ; max temp to set this state
DnSetMaxTemp        .equ    155d        ; max temp to set this state
Version             .equ    56H         ; set the version number

check_sum_value     .equ    05AH
TIMER_0             .EQU    10H
TIMER_0_EN          .EQU    03H
TIMER_1_EN          .EQU    0CH

MOTOR_HI            .EQU    034H
MOTOR_LO            .EQU    0BCH
LIGHT               .EQU    0FFH
LIGHT_ON            .EQU    02H
MOTOR_UP            .EQU    01H
MOTOR_DN            .EQU    04H
DN_LIMIT            .EQU    02H
UP_LIMIT            .EQU    01H
DIS_SW              .EQU    10000000B
CDIS_SW             .EQU    01111111B
SWITCHES            .EQU    01000000B
CHARGE_SW           .EQU    00100000B
CCHARGE_SW          .EQU    11011111B
COMPARATORS         .EQU    30H
DOWN_COMP           .EQU    20H
UP_COMP             .EQU    10H
P01M_INIT           .EQU    01000100B   ; set mode p00-p03 out p04-p07in
P2M_INIT            .EQU    11100000B
P3M_INIT            .EQU    00000001B   ; set port3 p30-p33 input DIGITAL mode
P01S_INIT           .EQU    00000010B
P2S_INIT            .EQU    10000010B
P3S_INIT            .EQU    10100000B


FLASH               .EQU    0FFH
WORKLIGHT           .EQU    02H

COM_CHARGE          .EQU    2
WORK_CHARGE         .EQU    20
VAC_CHARGE          .EQU    80

COM_DIS             .EQU    01
WORK_DIS            .EQU    04
VAC_DIS             .EQU    24

CMD_TEST            .EQU    00
WL_TEST             .EQU    01
```

```
VAC_TEST          .EQU   02
CHARGE            .EQU   03

AUTO_REV          .EQU   00H
UP_DIRECTION      .EQU   01H
UP_POSITION       .EQU   02H
DN_DIRECTION      .EQU   04H
DN_POSITION       .EQU   05H
STOP              .EQU   06H
CMD_SW            .EQU   01H
LIGHT_SW          .EQU   02H
VAC_SW            .EQU   04H


;------------------------------------------------------------------
;         PERIODS
;------------------------------------------------------------------

AUTO_HI           .EQU   00H          ; auto rev timer .5 sec
AUTO_LO           .EQU   0F4H
FLASH_HI          .EQU   00H          ; .25 sec flash
FLASH_LO          .EQU   07AH
SET_TIME_HI       .EQU   02H          ; 4.5 MIN
SET_TIME_LO       .EQU   02H          ; 4.5 MIN
SET_TIME_PRE      .EQU   0FBH         ; 4.5 MIN
ONE_SEC           .EQU   0F4H         ; WITH A /2 IN FRONT
SwPeriod          .equ   150d         ; switch period = 300uS
RsPeriod          .equ   104d         ; RS232 period 2400 Baud 208uS


CMD_MAKE          .EQU   8D           ; cycle count *10mS
CMD_BREAK         .EQU   (255D-8D)
LIGHT_MAKE        .EQU   8D           ; cycle count *11mS
LIGHT_BREAK       .EQU   (255D-8D)
VAC_MAKE_OUT      .EQU   4D           ; cycle count *100mS
VAC_BREAK_OUT     .EQU   (255D-4D)
VAC_MAKE_IN       .EQU   2D
VAC_BREAK_IN      .EQU   (255D-2D)

VAC_DEL           .EQU   8D
CMD_DEL_EX        .EQU   4D
VAC_DEL_EX        .EQU   50D


;------------------------------------------------------------------
;         ADDRESSES
;------------------------------------------------------------------

AddressA0         .equ   00H
AddressA1         .equ   02H
AddressA2         .equ   04H
AddressA3         .equ   06H
AddressA4         .equ   08H
AddressA5         .equ   0AH
```

```
AddressA6               .equ    0CH
AddressA7               .equ    0EH
AddressA8               .equ    10H
AddressA9               .equ    12H
AddressA10              .equ    14H
AddressA11              .equ    16H
AddressB                .equ    18H
AddressC                .equ    1AH
AddressCounter          .equ    1CH
AddressVacation         .equ    1EH
AddressApointer         .equ    1FH
AddressUpForceTable     .equ    20H
AddressTemperature      .equ    2EH
AddressUpLimit          .equ    2FH
AddressDownForceTable   .equ    30H
AddressLastOperation    .equ    3EH
AddressDownLimit        .equ    3FH


        .IF     E21
ALL_ON_IMR              .equ    00111111b       ; turn on  int for timers rpm auxobs
RadioOffIMR             .equ    00111100B       ; turn radio off durring autolearn cycle
RETURN_IMR              .equ    00111111b       ; return on the IMR
        .ELSE
ALL_ON_IMR              .equ    00111101b       ; turn on  int for timers rpm auxobs
RadioOffIMR             .equ    00111100B       ; turn radio off durring autolearn cycle
RETURN_IMR              .equ    00111101b       ; return on the IMR
        .ENDIF

;-----------------------------------------------------------------------
;       GLOBAL REGISTERS
;-----------------------------------------------------------------------

STATUS                  .EQU    04H
STATE                   .EQU    05H             ; state register
FORCE_PRE               .EQU    06H
FORCE_IGNORE            .EQU    07H
AUTO_DELAY_HI           .EQU    08H
AUTO_DELAY_LO           .EQU    09H
AUTO_DELAY              .EQU    08H
MOTOR_TIMER_HI          .EQU    0AH
MOTOR_TIMER_LO          .EQU    0BH
MOTOR_TIMER             .EQU    0AH
LIGHT_TIMER_HI          .EQU    0CH
LIGHT_TIMER_LO          .EQU    0DH
LIGHT_TIMER             .EQU    0CH
FourDFlag               .equ    0EH
PRE_LIGHT               .EQU    0FH


TIMER_GROUP             .EQU    10H
rsrto                   .equ    r5
obs_flag                .equ    r6
rs232do                 .equ    r7
rs232di                 .equ    r8
rscommand               .equ    r9
```

```
rs_temp_hi          .equ    r10
rs_temp_lo          .equ    r11
rs_temp             .equ    rr10
rs232docount        .equ    r10
rs232dicount        .equ    r11
rs232odelay         .equ    r12
rs232idelay         .equ    r13
rs232page           .equ    r15

VACCHANGE           .EQU    TIMER_GROUP+0
VACFLASH            .EQU    TIMER_GROUP+1
VACFLAG             .EQU    TIMER_GROUP+2
FAULT               .EQU    TIMER_GROUP+3
R_DEAD_TIME         .EQU    TIMER_GROUP+4
RsRto               .EQU    TIMER_GROUP+5
OBS_FLAG            .EQU    TIMER_GROUP+6
RS232DO             .EQU    TIMER_GROUP+7
RS232DI             .EQU    TIMER_GROUP+8
RSCOMMAND           .EQU    TIMER_GROUP+9
RS232DOCOUNT        .EQU    TIMER_GROUP+10
RS232DICOUNT        .EQU    TIMER_GROUP+11
RS232ODELAY         .EQU    TIMER_GROUP+12
RS232IDELAY         .EQU    TIMER_GROUP+13
Jog                 .EQU    TIMER_GROUP+14
RS232PAGE           .EQU    TIMER_GROUP+15

;***************************************************************
;
; LEARN EE GROUP FOR LOOPS ECT
;***************************************************************
;
LEARNEE_GRP         .equ    20H                 ;
RADIO_CMD           .equ    LEARNEE_GRP         ;
RSSTART             .equ    LEARNEE_GRP+1       ;
TEMP                .equ    LEARNEE_GRP+2       ;
LEARNDB             .equ    LEARNEE_GRP+3       ; learn debouncer
LEARNT              .equ    LEARNEE_GRP+4       ; learn timer
ERASET              .equ    LEARNEE_GRP+5       ; erase timer
MTEMPH              .equ    LEARNEE_GRP+6       ; memory temp
MTEMPL              .equ    LEARNEE_GRP+7       ; memory temp
MTEMP               .equ    LEARNEE_GRP+8       ; memory temp
SERIAL              .equ    LEARNEE_GRP+9       ; serial data to and from nonvol memory
ADDRESS             .equ    LEARNEE_GRP+10      ; address for the serial nonvol memory
T0EXT               .equ    LEARNEE_GRP+11      ; timer 0 extend dec every T0 int
RSCCOUNT            .equ    LEARNEE_GRP+12      ;
T125MS              .equ    LEARNEE_GRP+13      ; 125mS counter
OnePass             .equ    LEARNEE_GRP+14      ;
SKIPRADIO           .equ    LEARNEE_GRP+15      ; flag to skip the radio read and write if
                                                ; learn or vacation are talking to it

temp                .equ    r2                  ;
learndb             .equ    r3                  ; learn debouncer
learnt              .equ    r4                  ; learn timer
eraset              .equ    r5                  ; erase timer
mtemph              .equ    r6                  ; memory temp
mtempl              .equ    r7                  ; memory temp
mtemp               .equ    r8                  ; memory temp
serial              .equ    r9                  ; serial data to and from nonvol memory
```

```
address         .equ    r10            ; address for the serial nonvol memory
t0ext           .equ    r11            ; timer 0 extend dec every T0 int
t125ms          .equ    r13            ; 125mS counter
skipradio       .equ    r15            ; flag to skip the radio read and write if
                                       ; learn or vacation are talking to it


RPM_GROUP       .EQU    30H

stackreason     .equ    r0
stackflag       .equ    r1
rpm_temp_hi     .equ    r2
rpm_temp_lo     .equ    r3
rpm_temp        .equ    rr2
rpm_past_hi     .equ    r4
rpm_past_lo     .equ    r5
rpm_past        .equ    rr4
rpm_period_hi   .equ    r6
rpm_period_lo   .equ    r7
rpm_period      .equ    rr6
rpm_count       .equ    r8
rpm_diff_hi     .equ    r9
rpm_diff_lo     .equ    r10
rpm_2past_hi    .equ    r11
rpm_2past_lo    .equ    r12
rpm_time_out    .equ    r15

STACKREASON     .EQU    RPM_GROUP+0
STACKFLAG       .EQU    RPM_GROUP+1
RPM_TEMP_HI     .EQU    RPM_GROUP+2
RPM_TEMP_LO     .EQU    RPM_GROUP+3
RPM_PAST_HI     .EQU    RPM_GROUP+4
RPM_PAST_LO     .EQU    RPM_GROUP+5
RPM_PERIOD_HI   .EQU    RPM_GROUP+6
RPM_PERIOD_LO   .EQU    RPM_GROUP+7
RPM_COUNT       .EQU    RPM_GROUP+8
RPM_DIFF_HI     .EQU    RPM_GROUP+9
RPM_DIFF_LO     .EQU    RPM_GROUP+10
RPM_2PAST_HI    .EQU    RPM_GROUP+11
RPM_2PAST_LO    .EQU    RPM_GROUP+12
MinTimer        .EQU    RPM_GROUP+13
TDifference     .EQU    RPM_GROUP+14
RPM_TIME_OUT    .EQU    RPM_GROUP+15


;*******************************************************************
; RADIO GROUP
;*******************************************************************
RADIO_GRP       .equ    40H            ;
RTEMP           .equ    RADIO_GRP      ; radio temp storage
RTEMPH          .equ    RADIO_GRP+1    ; radio temp storage high
RTEMPL          .equ    RADIO_GRP+2    ; radio temp storage low
RTIMEAH         .equ    RADIO_GRP+3    ; radio active time high byte
```

```
RTIMEAL         .equ    RADIO_GRP+4        ; radio active time low byte
RTIMEIH         .equ    RADIO_GRP+5        ; radio inactive time high byte
RTIMEIL         .equ    RADIO_GRP+6        ; radio inactive time low byte
RTIMEPH         .equ    RADIO_GRP+7        ; radio past time high byte
RTIMEPL         .equ    RADIO_GRP+8        ; radio past time low byte
RADIO3H         .equ    RADIO_GRP+9        ; 3 mS code storage high byte
RADIO3L         .equ    RADIO_GRP+10       ; 3 mS code storage low byte
RADIO1H         .equ    RADIO_GRP+11       ; 1 mS code storage high byte
RADIO1L         .equ    RADIO_GRP+12       ; 1 mS code storage low byte
RADIOC          .equ    RADIO_GRP+13       ; radio word count
RTIMEDH         .equ    RADIO_GRP+14       ; radio difference of active and inactive
RTIMEDL         .equ    RADIO_GRP+15       ; radio difference
rtemp           .equ    r0                 ; radio temp storage
rtemph          .equ    r1                 ; radio temp storage high
rtempl          .equ    r2                 ; radio temp storage low
rtimeah         .equ    r3                 ; radio active time high byte
rtimeal         .equ    r4                 ; radio active time low byte
rtimeih         .equ    r5                 ; radio inactive time high byte
rtimeil         .equ    r6                 ; radio inactive time low byte
rtimeph         .equ    r7                 ; radio past time high byte
rtimepl         .equ    r8                 ; radio past time low byte
radio3h         .equ    r9                 ; 3 mS code storage high byte
radio3l         .equ    r10                ; 3 mS code storage low byte
radio1h         .equ    r11                ; 1 mS code storage high byte
radio1l         .equ    r12                ; 1 mS code storage low byte
radioc          .equ    r13                ; radio word count
rtimedh         .equ    r14                ; radio difference of active and inactive
rtimedl         .equ    r15                ; radio difference

ForceTable1     .equ    50H

Force0Hi        .equ    ForceTable1+0      ; force at the bottom of the door
Force0Lo        .equ    ForceTable1+1      ;
Force1Hi        .equ    ForceTable1+2      ;
Force1Lo        .equ    ForceTable1+3      ;
Force2Hi        .equ    ForceTable1+4      ;
Force2Lo        .equ    ForceTable1+5      ;
Force3Hi        .equ    ForceTable1+6      ;
Force3Lo        .equ    ForceTable1+7      ;
Force4Hi        .equ    ForceTable1+8      ;
Force4Lo        .equ    ForceTable1+9      ;
Force5Hi        .equ    ForceTable1+10     ;
Force5Lo        .equ    ForceTable1+11     ;
Force6Hi        .equ    ForceTable1+12     ; force at the worst case top
Force6Lo        .equ    ForceTable1+13     ;
Force7Hi        .equ    ForceTable1+14     ;
Force7Lo        .equ    ForceTable1+15     ; force address pointer


ForceTable2     .equ    60H

Force8Hi        .equ    ForceTable2+0      ; force at the bottom of the door
Force8Lo        .equ    ForceTable2+1      ;
Force9Hi        .equ    ForceTable2+2      ;
Force9Lo        .equ    ForceTable2+3      ;
Force10Hi       .equ    ForceTable2+4      ;
```

```
Force10Lo       .equ    ForceTable2+5     ;
Force11Hi       .equ    ForceTable2+6     ;
Force11Lo       .equ    ForceTable2+7     ;
Force12Hi       .equ    ForceTable2+8     ;
Force12Lo       .equ    ForceTable2+9     ;
Force13Hi       .equ    ForceTable2+10    ;
Force13Lo       .equ    ForceTable2+11    ;
Force14Hi       .equ    ForceTable2+12    ; force at the worst case top
Force14Lo       .equ    ForceTable2+13    ;
ForceTemp       .equ    ForceTable2+14    ;
ForceAddress    .equ    ForceTable2+15    ; force address pointer

forcetemp       .equ    r14
forceaddress    .equ    r15



FORCE_GRP       .equ    70H
CHECK_GRP       .equ    70H
check_sum       .equ    r0                ; check sum pointer
rom_data        .equ    r1
test_adr_hi     .equ    r2
test_adr_lo     .equ    r3
test_adr        .equ    rr2

forces          .equ    r0                ;
up_force_hi     .equ    r1                ;
up_force_lo     .equ    r2                ;
dn_force_hi     .equ    r3                ;
dn_force_lo     .equ    r4                ;
position_hi     .equ    r11               ;
position_lo     .equ    r12               ;
l_a_c           .equ    r14               ;

CHECK_SUM       .equ    CHECK_GRP+0       ; check sum reg for por
ROM_DATA        .equ    CHECK_GRP+1       ; data read

FORCES          .equ    FORCE_GRP         ; force max during setting
                                          ; 3 = MAX force 10mS
                                          ; 2 = HI force 9 mS
                                          ; 1 = MID force 8.25 mS
                                          ; else = LOW force 7.75 mS

UP_FORCE_HI     .equ    FORCE_GRP+1       ;
UP_FORCE_LO     .equ    FORCE_GRP+2       ;
DN_FORCE_HI     .equ    FORCE_GRP+3       ;
DN_FORCE_LO     .equ    FORCE_GRP+4       ;
AOBSF           .equ    FORCE_GRP+5       ;
FAULTCODE       .equ    FORCE_GRP+6       ;
AOBSTEST        .equ    FORCE_GRP+7       ;
FAULTTIME       .equ    FORCE_GRP+8       ;
RPM_ACOUNT      .equ    FORCE_GRP+9       ;
UpDown          .equ    FORCE_GRP+10      ; up to down direction change timer
```

```
POSITION_HI      .equ    FORCE_GRP+11      ;
POSITION_LO      .equ    FORCE_GRP+12      ;
P5UTD            .equ    FORCE_GRP+13      ;
L_A_C            .equ    FORCE_GRP+14      ; limits are changing
AOBS_FLAG        .equ    FORCE_GRP+15      ; flag for pass point

PRADIO_GRP       .equ    80H
SDISABLE         .equ    PRADIO_GRP+0      ; system disable timer
PRADIO3H         .equ    PRADIO_GRP+1      ; 3 mS code storage high byte
PRADIO3L         .equ    PRADIO_GRP+2      ; 3 mS code storage low byte
PRADIO1H         .equ    PRADIO_GRP+3      ; 1 mS code storage high byte
PRADIO1L         .equ    PRADIO_GRP+4      ; 1 mS code storage low byte
RTO              .equ    PRADIO_GRP+5      ; radio time out
RFLAG            .equ    PRADIO_GRP+6      ; radio flags
RINFILTER        .equ    PRADIO_GRP+7      ; radio input filter
LIGHT1S          .equ    PRADIO_GRP+8      ; light timer for 1second flash
DOG2             .equ    PRADIO_GRP+9      ; second watchdog
GotSwitch        .equ    PRADIO_GRP+0AH    ; found a switch set
FAULTFLAG        .equ    PRADIO_GRP+0BH    ; flag for fault blink stops radio blink
MOTDEL           .equ    PRADIO_GRP+0CH    ; motor time delay
LIGHTS           .equ    PRADIO_GRP+0DH    ; light state
DELAYC           .equ    PRADIO_GRP+0EH    ; for the time delay for command
WIN_FLAG         .equ    PRADIO_GRP+0FH    ; flag for the operation of the window
                                           ; for the pass point
                                           ; 0 = skip pass point window
                                           ; not 0 do pass point


FORCE2_GRP       .equ    090H
MAX_F_HI         .equ    FORCE2_GRP        ; temp storage for the max force reading
MAX_F_LO         .equ    FORCE2_GRP+1
P32_MAX_HI       .equ    FORCE2_GRP+2      ; delayed storage every 32 steps
P32_MAX_LO       .equ    FORCE2_GRP+3
AOBSRPM          .equ    FORCE2_GRP+4      ; the count of rpm pulses from aobs
UP_LIM_HI        .equ    FORCE2_GRP+5      ; the up limit count
UP_LIM_LO        .equ    FORCE2_GRP+6      ; the up limit count
DN_LIM_HI        .equ    FORCE2_GRP+7      ; the down limit count
DN_LIM_LO        .equ    FORCE2_GRP+8      ; the down limit count
AOBSB            .equ    FORCE2_GRP+9      ; the RPM count of the protector break
AOBSNB           .equ    FORCE2_GRP+10     ; the RPM count of protector make
AOBSSTATUS       .equ    FORCE2_GRP+11     ; the protector sta :: 00 beam mace
                                           ; FF beam broken
AOBSSTATE        .equ    FORCE2_GRP+12     ; the state of the zero point test
                                           ; 00 = waiting for first block
                                           ; 01 = blocked < 12 counts
                                           ;          clear unblocked
                                           ; 02 = waiting for unblocked
                                           ;          (is blocked > 30)
                                           ; 03 = unblocked < 12 counts
                                           ;          clear blocked
                                           ; 04 = waiting for blocked
                                           ;          (is unblocked > 30)
                                           ; 05 = blocked < 12 counts
                                           ;          clear unblocked
                                           ; 06 = waiting for unblocked
                                           ;          (is blocked > 30)
```

```
                                                          ; 07 = zero clear AOBSRPM
                                                          ;         clear AOBSSTATE
PWINDOW             .equ      FORCE2_GRP+13               ; window
RsTimer             .equ      FORCE2_GRP+14               ; RS232 operation timer 4 S inc till FF
                                                          ; FF = RS232 off switches operational
                                                          ; else RS232 on switches
T1Mirror            .equ      FORCE2_GRP+15               ; T1 setting mirror


DB_GROUP            .EQU      0A0H
SW_DATA             .EQU      DB_GROUP
ONEP2               .EQU      DB_GROUP+1                  ; 1.2 SEC TIMER TICK .125
LAST_CMD            .EQU      DB_GROUP+2                  ; LAST COMMAND FROM
                                                          ; = 55 WALL CONTROL
                                                          ; = 00 RADIO
                                                          ; = AA RS232
BCODEFLAG           .EQU      DB_GROUP+3                  ; B CODE FLAG
                                                          ; 77 = b code
RPMONES             .EQU      DB_GROUP+4                  ; RPM PULSE ONE SECOND DISABLE
RPMCLEAR            .EQU      DB_GROUP+5                  ; RPM PULSE CLEAR ,TEST TIMER
FAREVFLAG           .EQU      DB_GROUP+6                  ; RPM FORCED AREV FLAG
                                                          ; 88H FOR A FORCED REVERSE

FLASH_FLAG          .EQU      DB_GROUP+7
FLASH_DELAY_HI      .EQU      DB_GROUP+8
FLASH_DELAY_LO      .EQU      DB_GROUP+9
FLASH_DELAY         .EQU      DB_GROUP+8
FLASH_COUNTER       .EQU      DB_GROUP+0AH
REASON              .EQU      DB_GROUP+0BH
                                                          ; 00    COMMAND
                                                          ; 10    RADIO COMMAND
                                                          ; 20    FORCE
                                                          ; 30    AUXOBS
                                                          ; 40    AUTOREVERSE TIMEOUT
                                                          ; 50    LIMIT
                                                          ; 60    EARLY LIMIT
                                                          ; 70    MOTOR MAX TIME OUT
                                                          ; 80    FORCED AREV FROM RPM
                                                          ; 90    CLOSED COMMAND HELD
                                                          ; A0    CLOSED WITH RADIO HELD
                                                          ; F0    No pass point
LIGHT_FLAG          .EQU      DB_GROUP+0CH
CMD_DEB             .EQU      DB_GROUP+0DH
LIGHT_DEB           .EQU      DB_GROUP+0EH
VAC_DEB             .EQU      DB_GROUP+0FH


BACKUP_GRP          .equ      0B0H
LearnLed            .equ      BACKUP_GRP+0
                                                          ; led control
                                                          ; 00XX XXXX = Led Blink from radio
                                                          ; 01XX XXXX = Blink From Fault
                                                          ; 10XX XXXX = Learn mode
                                                          ; XXFF FFFF = off
```

```
                                                     ; XXNN NNNN count at 3mS rate
RsMode            .equ    BACKUP_GRP+1               ; = 232D if RS232 only set from U code
ForceAddHi        .equ    BACKUP_GRP+2               ; force adder From temperature
ForceAddLo        .equ    BACKUP_GRP+3
ForceAdd          .equ    BACKUP_GRP+2
MotorTempHi       .equ    BACKUP_GRP+4
MotorTempLo       .equ    BACKUP_GRP+5
MotorTemp         .equ    BACKUP_GRP+4
Temperature       .equ    BACKUP_GRP+6
P8Counter         .equ    BACKUP_GRP+7
PastTemp          .equ    BACKUP_GRP+8
BRPM_TIME_OUT     .equ    BACKUP_GRP+9
BFORCE_IGNORE     .equ    BACKUP_GRP+0AH
BSTATE            .equ    BACKUP_GRP+0BH
BAUTO_DELAY_HI    .equ    BACKUP_GRP+0CH
BAUTO_DELAY_LO    .equ    BACKUP_GRP+0DH
BAUTO_DELAY       .equ    BACKUP_GRP+0CH
BCMD_DEB          .equ    BACKUP_GRP+0FH

STACKTOP          .equ    238                        ; start of the stack
STACKEND          .equ    0C0H                       ; end of the stack

RS232OS           .equ    00010000B                  ; RS232 output bit set
RS232OC           .equ    11101111B                  ; RS232 output bit clear
RS232OP           .equ    P3                         ; RS232 output port

RS232IP           .equ    P0                         ; RS232 input port
RS232IM           .equ    01000000B                  ; RS232 mask

RsInputModeAnd    .equ    10101111B                  ;
RsInputModeOr     .equ    10100000B                  ;

RsOutputModeAnd   .equ    10101111B                  ;
RsOutputModeOr    .equ    10100000B                  ;

csh               .equ    00010000B                  ; chip select high for the 93c46
csl               .equ    11101111B                  ; chip select low for 93c46
clockh            .equ    00001000B                  ; clock high for 93c46
clockl            .equ    11110111B                  ; clock low for 93c46
doh               .equ    00000100B                  ; data out high for 93c46
dol               .equ    11111011B                  ; data out low for 93c46
psmask            .equ    01000000B                  ; mask for the program switch
csport            .equ    P2                         ; chip select port
dioport           .equ    P2                         ; data i/o port
clkport           .equ    P2                         ; clock port
psport            .equ    P2                         ; program switch port

WATCHDOG_GROUP .EQU       0FH
pcon              .equ    r0
smr               .equ    r11
wdtmr             .equ    r15
```

14003305.020402

```
WDT             .macro
                .byte   5fh
                .endm

FILL            .macro
                .byte   0FFh
                .endm

TRAP            .macro
                jp      start
                jp      start
                jp      start
                jp      start
                jp      start
                .endm
TRAP10          .macro
                TRAP
                TRAP
                TRAP
                TRAP
                TRAP
                TRAP
                TRAP
                TRAP
                TRAP
                TRAP
                .endm

;**************************************************************
;*
;*          Interrupt Vector Table
;*
;**************************************************************
        .IF E21

                .org    0000H

                .word   RADIO_INT               ;IRQ0. P3.2
                .word   RADIO_INT               ;IRQ1, P3.3
                .word   AUX_OBS                 ;IRQ2, P3.1
                .word   RPM                     ;IRQ3, P3.0
                .word   Timer1Int               ;IRQ4, T0
                .word   Timer2Int               ;IRQ5, T1

        .ELSE
                .org    0000H

                .word   RADIO_INT               ;IRQ0. P3.2
                .word   000CH                   ;IRQ1, P3.3
                .word   RPM                     ;IRQ2. P3.1
                .word   AUX_OBS                 ;IRQ3. P3.0
                .word   Timer1Int               ;IRQ4. T0
                .word   Timer2Int               ;IRQ5. T1
        .ENDIF
```

```
.page

        .org    000CH

  jp    START                                   ; start jmps to start at location 0101

;-----------------------------------------------------------------
;       RS232 DATA ROUTINES
;-----------------------------------------------------------------

;       enter rs232 start with word to output in rs232do
RS232OSTART:
                                                ; set the Output mode
        or      RS232OP,#RsOutputModeOr         ;
        and     RS232OP,#RsOutputModeAnd        ;
        push    rp                              ; save the rp
        srp     #TIMER_GROUP                    ; set the group pointer
        cp      rs232odelay,#00H                ; test for ready
        jr      z,RsReady

        djnz    rs232odelay,NORSIN
RsReady:

        clr     RSSTART                         ; one shot
        ld      rs232odelay,#04                 ; set the period

        clr     rs232docount                    ; start with the counter at 0
        or      RS232OP,#RS232OS                ; set the output
        jr      NORSIN                          ;


RS232:
        cp      RSSTART,#0FFH                   ; test for the start flag
        jr      z,RS232OSTART
RS232OUTPUT:
        push    rp                              ; save the rp
        srp     #TIMER_GROUP                    ; set the group pointer
        cp      rs232docount,#11d               ; test for last
        jr      ult,RS232R
        jr      ugt,InputMode

        and     RS232OP,#RS232OC                ; clear the output
        inc     rs232docount                    ; one shot

InputMode:                                      ; set the input mode
        or      RS232OP,#RsInputModeOr          ;
        and     RS232OP,#RsInputModeAnd         ;

        JR      NORSOUT
RS232R:
        ld      rs232dicount,#0F0H              ; set a time delay
        djnz    rs232odelay,NORSIN              ; cycle count time delay
        inc     rs232docount                    ; set the count for the next cycle
        scf                                     ; set the carry flag for stop bits
        rrc     rs232do                         ; get the data into the carry
```

```
        jr      c,RS232SET              ; if the bit is high then set

        or      RS232OP,#RS232OS        ; set the output
        jr      SETTIME                 ; find the delay time
RS232SET:
        and     RS232OP,#RS232OC        ; clear the output
SETTIME:
        ld      rs232odelay,#4d         ; set the data output delay
        jr      NORSIN

NORSOUT:
RS232INPUT:

        cp      rs232dicount,#0FFH      ; test  mode
        jr      nz,RECEIVING            ; if receiving then jump
        tm      RS232IP,#RS232IM        ; test the incoming data

        jr      nz,NORSIN               ; if the line is still idle then skip

        clr     rs232dicount            ; start at 0
        ld      rs232idelay,#2d         ; set the delay to 1/2
RECEIVING:
        djnz    rs232idelay,NORSIN      ; skip till delay is up
        inc     rs232dicount            ; bit counter
        cp      rs232dicount,#10d       ; test for last timeout
        jr      z,DIEVEN
        tm      RS232IP,#RS232IM        ; test the incoming data
        rcf                             ; clear the carry

        jr      z,SKIPSETTING           ; if input bit not set skip setting carry
        scf                             ; set the carry
SKIPSETTING:
        rrc     rs232di                 ; save the data into the memory
        ld      rs232idelay,#4d         ; set the delay
        jr      NORSIN
DIEVEN:
        ld      rs232dicount,#0FFH      ; turn off the input till next start
        ld      rscommand,rs232di       ; save the value
        clr     RSCCOUNT                ; clear the counter
NORSIN:
        pop     rp                      ; return the rp
        ret


                .org    101H            ; start address

;*******************************************************************
; REGISTER INITILIZATION
;*******************************************************************

start.
START:
        di                              ; turn off the interrupt for init
        .IF     E21
        xor     P1,#00000001B           ; Kick the external dog
        .ELSE
```

```
        ld      RP,#WATCHDOG_GROUP
        ld      wdtmr,#00001111B                    ; rc dog 100mS
        WDT                                         ; kick the dog
        .ENDIF
        clr     RP                                  ; clear the register pointer

;*************************************************************
;*          Internal RAM Test and Reset All RAM =    mS
;*************************************************************
        srp     #0F0h                               ; point to control group use stack
        ld      r15,#4                              ;r15= pointer (minimum of RAM)
write_again:
        .IF     E21
        xor     P1,#00000001B                       ; Kick the external dog
        .ELSE
        WDT                                         ; KICK THE DOG
        .ENDIF
        ld      r14,#1
write_again1:
        ld      @r15,r14                            ;write 1,2,4,8,10,20,40,80
        cp      r14,@r15                            ;then compare
        jr      ne,system_error
        rl      r14
        jr      nc,write_again1
        clr     @r15                                ;write RAM(r5)=0 to memory
        inc     r15
        cp      r15,#240
        jr      ult,write_again

;*************************************************************
; STACK INITILIZATION
;*************************************************************
STACK:
        clr     254
        ld      255,#238D                           ; set the start of the stack
        ld      P0,#P01S_INIT                       ; RESET all ports
        ld      P2,#P2S_INIT                        ;
        ld      P3,#P3S_INIT                        ;
        ld      P01M,#P01M_INIT                     ; set mode p00-p03 out p04-p07in
        ld      P3M,#P3M_INIT                       ; set port3 p30-p33 input analog mode
                                                    ; p34-p37 outputs
        ld      P2M,#(P2M_INIT+0)                   ; set port 2 mode

;*************************************************************
;*                 Checksum Test
;*************************************************************
CHECKSUMTEST:
        srp     #CHECK_GRP
        ld      test_adr_hi,#0FH
        ld      test_adr_lo,#0FFH                   ;maximum address=fffh
add_sum:
        .IF     E21
        xor     P1,#00000001B                       ; Kick the external dog
```

```
        .ELSE
        WDT                                     ; KICK THE DOG
        .ENDIF
        call    PORTINIT                        ; port initilization
        ldc     rom_data.@test_adr              ;read ROM code one by one
        add     check_sum.rom_data              ;add it to checksum register
        decw    test_adr                        ;increment ROM address
        jr      nz.add_sum                      ;address=0 ?
        cp      check_sum.#check_sum_value
        jr      system_ok                       ;temp test
        jr      z.system_ok                     ;check final checksum = 00 ?

system_error:
        and     P3,#00111111B                   ; turn off both outputs
        or      P3,#01000000B                   ; turn on the led
        jr      system_error

        .byte   256-check_sum_value
system_ok:

        .IF     E21
        xor     P1,#00000001B                   ; Kick the external dog
        .ELSE
        WDT                                     ; KICK THE DOG
        .ENDIF

        ld      STACKEND,#STACKTOP              ; start at the top of the stack
SETSTACKLOOP:
        ld      @STACKEND,#01H                  ; set the value for the stack vector
        dec     STACKEND                        ; next address
        cp      STACKEND.#STACKEND              ; test for the last address
        jr      nz.SETSTACKLOOP                 ; loop till done

CLEARDONE:

        ld      STATE,#05d                      ; set the state to DOWN POSITION
        ld      BSTATE,#05d                     ; FORCING UP TRAVEL FIRST STEP
        ld      LIGHT_TIMER_HI,#SET_TIME_HI     ; set the light period
        ld      LIGHT_TIMER_LO,#SET_TIME_LO     ; for the 4.5 min timer
        ld      PRE_LIGHT,#SET_TIME_PRE         ;
        ld      CMD_DEB,#0FFH                   ; in case of shorted switches
        ld      BCMD_DEB.#0FFH                  ; in case of shorted switches
        ld      VAC_DEB,#0FFH                   ;
        ld      LIGHT_DEB.#0FFH                 ;
        ld      ERASET,#0FFH                    ; set the erase timer
        ld      LEARNDB.#0FFH                   ; set the learn debouncer
        ld      LEARNT.#0FFH                    ; set the learn timer
        ld      RTO,#0FFH                       ; set the radio time out
        ld      RS232DOCOUNT.#012d              ;
        ld      RPMONES,#244d                   ; set the hold off

;************************************************************
; TIMER INITILIZATION
;************************************************************
```

```
TIMER:
        ld      PRE0,#00001001B             ; set the prescaler to / 2 for 8Mhz
        ld      T0,#000H                    ; set the counter to count FF through 0
        ld      PRE1,#00001011B             ; set the prescaler to / 2 for 8Mhz
        ld      T1Mirror,#SwPeriod          ; set the period to 300uS for switches
        ld      T1,T1Mirror                 ;
        ld      TMR,#00001111B              ; turn on the timer
        call    PORTINIT                    ; init the ports

;*******************************************************************
; SET PORTS AND DIVIDER
;*******************************************************************
        .IF     E21

        .ELSE
        ld      RP,#WATCHDOG_GROUP
        ld      smr,#00100010B              ; reset the xtal / number
        ld      pcon,#01111110B             ; reset the pcon no comparator output
                                            ; no low emi mode
        .ENDIF
        ld      PRE0,#00001001B             ; set the prescaler to / 2 for 8Mhz

;*******************************************************************
; READ THE MEMORY AND GET THE VACFLAG
;*******************************************************************

        ld      SKIPRADIO,#0FFH             ;
        srp     #LEARNEE_GRP

        ld      address,#AddressVacation    ; set non vol address to the VAC flag
        call    READMEMORY                  ; read the value 2X 1X INIT
        call    READMEMORY                  ; read the value
        ld      VACFLAG,mtemph              ; read into volital

;*******************************************************************
; READ THE TEMPERATURE
;*******************************************************************

        clr     IMR                         ; turn off all interrupts
        ld      ADDRESS,#AddressTemperature ; read the motor temp from nonvol
        call    READMEMORY                  ; read the memory data
        clr     IMR                         ; turn off all interrupts
        ld      MotorTempHi,MTEMPH          ;
        ld      MotorTempLo,MTEMPL          ;
        call    TempMeasure                 ; read the temp

;*******************************************************************
; Reset the machine according to last state
;*******************************************************************
```

```
        ld      address,#AddressLastOperation
                                                    ; get the last operation
        call    READMEMORY                          ;

        ld      POSITION_HI,#07FH                   ; set the position to the temp
        ld      POSITION_LO,#0D4H                   ; limit till pass point
        ld      STATE,mtemph
        and     STATE,#00001111B                    ; remove the reason
        call    ReadLimits                          ; read the limits
        ld      ADDRESS,#AddressDownForceTable      ; point to the down force table
        cp      STATE,#5d                           ; test for the down limit
        jr      z,DownWake                          ; if so set the down limit
        cp      STATE,#2d                           ; test for at the up limit
        jr      z,UpWake                            ; if so then set the up limit
        jr      MidWake                             ; else in mid travel wake up
DownWake:
        ld      POSITION_HI,DN_LIM_HI               ; set the position as the down
        ld      POSITION_LO,DN_LIM_LO               ; limit
        inc     WIN_FLAG                            ; turn on the window
        jr      Wake                                ;
UpWake:
        ld      ADDRESS,#AddressUpForceTable        ; point to the down force table
        ld      POSITION_HI,UP_LIM_HI               ; set the position as the up
        ld      POSITION_LO,UP_LIM_LO               ; limit
        inc     WIN_FLAG                            ; turn on the window
        jr      Wake
MidWake:
        ld      STATE,#6d                           ; set the stopped state
        add     MotorTempHi,#T27Adder               ; increase temp


Wake:
        ld      BSTATE,STATE                        ; set the backup state
        call    ReadForceTable                      ; read the force table
        call    FIND_WINDOW                         ; find the window
        clr     SKIPRADIO


;*****************************************************************
; INITERRUPT INITILIZATION
;*****************************************************************

SETINTERRUPTS:
        .IF E21
        ld      IPR,#00101011B              ; set the priority to timer
        .ELSE
        ld      IPR,#00011010B              ; set the priority to timer
        .ENDIF
        ld      IMR,#ALL_ON_IMR             ; turn on the interrupt
        .IF     E21
        ld      IRQ,#00000000B              ; set the edge clear int
        .ELSE
        ld      IRQ,#01000000B              ; set the edge clear int
        .ENDIF
```

```
        ei                                      ; enable interrupt
;**************************************************************
; MAIN LOOP
;**************************************************************
MAINLOOP:
        clr     DOG2                            ; clear the second watchdog
        cp      Jog,#055H                       ; test for jog up
        jr      z,DoJogUp                       ;
        cp      Jog,#0AAH                       ; test for jog down
        jr      z,DoJogDn
        jr      JogSkip
DoJogUp:
        sub     UP_LIM_LO,#10d                  ; jog the limit
        sbc     UP_LIM_HI,#00H
        jr      JogMem

DoJogDn:
        add     UP_LIM_LO,#10d                  ; jog the limit
        adc     UP_LIM_HI,#00H

JogMem:
        clr     Jog                             ; one shot
        ld      SKIPRADIO,#0FFH                 ;
        ld      ADDRESS,#AddressUpLimit         ; set non vol address to the up limit
        ld      MTEMPH,UP_LIM_HI                ; save into nonvolital
        ld      MTEMPL,UP_LIM_LO                ;
        call    WRITEMEMORY                     ; write the value
        clr     SKIPRADIO                       ;
        ld      L_A_C,#30H                      ; set the jog operation

JogSkip:
        cp      OnePass,STATE                   ; test if read out of memory allready
        jr      z,SkipMemoryRead                ; if so then skip reading out of memory
        cp      L_A_C,#42H                      ; test if in learn mode
        jr      uge,LearnSkipMemoryRead         ; if so then skip reading out of memory
        cp      STATE,#1d                       ; test for the up state
        jr      z,UpTableRead                   ; if so read the up table
        cp      STATE,#4d                       ; test for the down state
        jr      z,DownTableRead                 ; if so read the down table
        jr      SkipMemoryRead                  ; else skip

DownTableRead:
        ld      SKIPRADIO,#0FFH                 ; turn off the radio read
        ld      ADDRESS,#AddressDownForceTable  ; read the down force table
        call    READMEMORY                      ; dummy read
        call    ReadForceTable                  ; read the force table
        clr     SKIPRADIO                       ; allow the radio function
        ld      OnePass,STATE                   ; save the state
        jr      SkipMemoryRead                  ;

UpTableRead:
        ld      OnePass,STATE                   ; save the state
        ld      SKIPRADIO,#0FFH                 ; turn off the radio read
```

```
        ld      ADDRESS,#AddressUpForceTable      ; read the up force table
        call    READMEMORY                        ; dummy read
        call    ReadForceTable                    ; read the force table
        clr     SKIPRADIO                         ; allow the radio function
        ld      OnePass,STATE                     ; save the state
        jr      SkipMemoryRead                    ;
LearnSkipMemoryRead:
        ld      OnePass,STATE                     ; save the state
SkipMemoryRead:
        cp      L_A_C,#42h                        ; test for in learn mode
        jr      uge,SkipReadForce                 ; if so then skip reading the force
        call    ReadForce                         ; read the present force value
SkipReadForce:

        call    PORTREF                           ; refresh the ports
        srp     #FORCE_GRP                        ; set the rp
        cp      l_a_c,#030H                       ; test for learn action
        jp      ult,CLRLAC                        ; if less then then clear number
        cp      l_a_c,#042H                       ; test for active learn limits
        jr      uge,LearnLimits
        cp      l_a_c,#32H                        ; test for the end of jog
        jp      ugt,CLRLAC                        ; if so then clear
        cp      l_a_c,#30H                        ; test for stop
        jp      z,G30
        cp      l_a_c,#31H                        ; test for start travel down
        jp      z,G31
        jp      G32                               ; else delay for up
LearnLimits:
        cp      l_a_c,#04Fh                       ; test for to large a number
        jp      z,STOREFL                         ; if = store the force and limits
        jp      ugt,CLRLAC                        ; if greater or = clear

        clr     WIN_FLAG                          ; turn off the window

        cp      l_a_c,#042H                       ; test for state 42
        jp      z,G42                             ; if so then stop motor and set force

        cp      l_a_c,#043H                       ; test for state 43
        jp      z,G43                             ; if so time delay then up

        cp      l_a_c,#044H                       ; test for state 44
        jp      z,G44                             ; if so travel up till cmd release

        cp      l_a_c,#045H                       ; test for state 45
        jp      z,G45                             ; if so clear timer set next state

        cp      l_a_c,#046H                       ; test for state 46
        jp      z,G46                             ; if so time delay then down

        cp      l_a_c,#04AH                       ; test for state 4A
        jp      z,G4A                             ; if so clear timer set next state

        cp      l_a_c,#04BH                       ; test for state 4B
        jp      z,G4B                             ; if so time delay then down

        cp      l_a_c,#04DH                       ; test for state 4D
```

```
        jp      z.G4D                           ; if so store the force table and
                                                ; set the up force table pointer

        jp      LACCS                           ; else exit


G42:
        inc     forces                          ; increase the forces
        cp      forces,#03                      ; test for the max setting
        jr      ule.SKIPFINC
        clr     forces                          ; reset if at the max
SKIPFINC:
        cp      forces,#03                      ; test for the max force
        jr      nz,FORCE2T                      ; if not then test for force 2 setting
FORCE3:
        ld      dn_force_lo,#088H               ; set the force to MAX
        ld      dn_force_hi,#013H

        jr      FORCESET
FORCE2T:
        cp      forces,#02                      ; test for the high force
        jr      nz,FORCE1T                      ; if not test for mid l
FORCE2:
        ld      dn_force_lo,#094H               ; set the force to HI
        ld      dn_force_hi,#011H

        jr      FORCESET
FORCE1T:
        CP      forces,#01                      ; test for mid low
        jr      nz,FORCE0                       ; IF NOT THE FORCE IS MIN
FORCE1:
        ld      dn_force_lo,#01DH               ; set the force to mid
        ld      dn_force_hi,#010H
        jr      FORCESET
FORCE0:
        ld      dn_force_lo,#023H               ; set the force to min
        ld      dn_force_hi,#00FH
        jr      FORCESET

FORCESET:
        ld      UP_FORCE_HI,dn_force_hi         ;
        ld      UP_FORCE_LO,up_force_lo         ;
        inc     L_A_C                           ; set the next state
        clr     P5UTD
        jp      LACCS
G30:
        cp      STATE,#DN_DIRECTION             ; test for traveling
        jr      z,Delay30
        cp      STATE,#UP_DIRECTION             ;
        jr      z.Delay30
        inc     L_A_C                           ; set the next state
        ld      P5UTD,#11d                      ; delay short
        jp      LACCS
Delay30:
        clr     P5UTD                           ; clear the timer
        call    SET_STOP_STATE                  ; stop the machine for .5 sec
```

```
        jp      LACCS                           ;
G31:
        cp      P5UTD.#012d                     ; test for the delay
        jp      nz,LACCS                        ; if not the skip
        clr     P5UTD                           ; clear the timer
        ld      LAST_CMD.#055H                  ; set the last command as wall cmd
        ld      SW_DATA.#CMD_SW                 ; set the switch data as command
        jp      LACCS                           ;
G32:
        cp      P5UTD.#012d                     ; test for the delay
        jp      nz.LACCS                        ; if not the skip
        clr     P5UTD                           ; clear the timer
        ld      LAST_CMD,#055H                  ; set the last command as wall cmd
        ld      SW_DATA.#CMD_SW                 ; set the switch data as command
        jp      LACCS                           ;

G43:
        cp      P5UTD.#06d                      ; test for the delay
        jp      nz,LACCS                        ; if not the skip
        call    SET_UP_DIR_STATE
        jp      LACCS                           ;
G44:
        cp      CMD_DEB.#0FFH                   ; test for the command being held
        jr      z.LACCS
        clr     FourDFlag                       ; clear the flag
        call    SET_UP_POS_STATE                ; set the up position state
        JR      LACCS
G45:
G4A:
        clr     P5UTD                           ; clear the timer
        inc     l_a_c
        jr      LACCS
G46:
        di
        clr     POSITION_HI                     ; clear the position
        clr     POSITION_LO                     ;
        ei
G4B:
        cp      P5UTD,#6d                       ; DELAY <.5 SECONDS
        jr      ne,LACCS                        ; if not just wait
        cp      l_a_c,#4BH                      ; test for set
        jr      nz,SkipDownInit
SetDownPointer:
        push    RP                              ; set the rp
        srp     #ForceTable2                    ;
        ld      forceaddress,#Force0Hi          ; set the address pointer to fill
        ld      forcetemp,#15d                  ; set the number of address
DownForceInit:
        ld      @forceaddress,DN_FORCE_HI       ; set the initial value
        inc     forceaddress
        ld      @forceaddress,DN_FORCE_LO
        inc     forceaddress
        djnz    forcetemp.DownForceInit         ; loop till done

        ld      forceaddress.POSITION_HI        ; get the position
        com     forceaddress                    ; turn it into the pointer
```

```
        inc     forceaddress                    ;
        cp      forceaddress,#0DH               ; test for the max
        jr      ult,Dn2X                        ; if not skip zeroing
        clr     forceaddress
Dn2X:
        rcf                                     ; *2
        rlc     forceaddress                    ;
        add     forceaddress,#Force0Hi          ;
        pop     RP
SkipDownInit:
        call    SET_DN_DIR_STATE
        jr      LACCS

G4D:
        cp      FourDFlag,#00                   ; test for 1 time only operation
        jr      nz,LACCS                        ; if not skip
        inc     FourDFlag                       ;

StoreDownForceTable:

        ld      Force0Hi,P32_MAX_HI             ; set the force to P32 for the reverse
        ld      Force0Lo,P32_MAX_LO             ;
        ld      ADDRESS,#AddressDownForceTable
        call    StoreForceTable


SetUpPointer:
        push    RP                              ; set the rp
        srp     #ForceTable2                    ;
        ld      forceaddress,#Force0Hi          ; set the address pointer to fill
        ld      forcetemp,#15d                  ; set the number of address
UpForceInit:
        ld      @forceaddress,UP_FORCE_HI       ; set the initial value
        inc     forceaddress
        ld      @forceaddress,UP_FORCE_LO
        inc     forceaddress
        djnz    forcetemp,UpForceInit           ; loop till done

        ld      forceaddress,#Force0Hi          ;
        pop     RP


        jr      LACCS                           ; exit


CLRLAC:
        clr     l_a_c                           ; clear the L_A_C reg
LACCSE:
        clr     P5UTD                           ; clear the timer for .5 reverse
LACCS:
        EI
        cp      VACCHANGE,#0AAH                 ; test for the vacation change flag
        jr      nz,NOVACCHG                     ; if no change the skip
        cp      VACFLAG,#0FFH                   ; test for in vacation
        jr      z,MCLEARVAC                     ; if in vac clear
```

```
        ld      VACFLAG,#0FFH                   ; set vacation
        jr      SETVACCHANGE                    ; set the change
MCLEARVAC:
        clr     VACFLAG                         ; clear vacation mode
SETVACCHANGE:
        clr     VACCHANGE                       ; one shot
        ld      SKIPRADIO,#0FFH                 ; set skip flag
        ld      ADDRESS,#AddressVacation        ; non vol address to the VAC flag
        ld      MTEMPH,VACFLAG                  ; store the vacation flag
        ld      MTEMPL,VACFLAG                  ;
        call    WRITEMEMORY                     ; write the value
        clr     SKIPRADIO                       ; clear skip flag
NOVACCHG:
        cp      STACKFLAG,#0AAH                 ; test for temperature storage
        jr      z,WriteTheTemperature           ; if so save it
        cp      STACKFLAG,#0FFH                 ; test for the change flag
        jr      nz,NOCHANGEST                   ; if no change skip updating

        srp     #LEARNEE_GRP                    ; set the register pointer
        clr     STACKFLAG                       ; clear the flag
        ld      SKIPRADIO,#0FFH                 ; set skip flag
        ld      address,#AddressCounter         ; set the non vol address to the cycle c
        call    READMEMORY                      ; read the value
        inc     mtempl                          ; increase the counter lower byte
        jr      nz,COUNTERDONE                  ;
        inc     mtemph                          ; increase the counter high byte
        jr      nz,COUNTERDONE                  ;
        call    WRITEMEMORY                     ; store the value
        inc     address                         ; get the next bytes
        call    READMEMORY                      ; read the data
        inc     mtempl                          ; increase the counter low byte
        jr      nz,COUNTERDONE                  ;
        inc     mtemph                          ; increase the vounter high byte
COUNTERDONE:
        call    WRITEMEMORY                     ; got the new address
CDONE:
        ld      address,#AddressLastOperation
        ld      mtemph,STACKREASON
        or      mtemph,STATE                    ; or in the state
        ld      mtempl,mtemph                   ; set both the same
        call    WRITEMEMORY                     ; write the value to stack
        clr     SKIPRADIO                       ; clear skip flag
WriteTheTemperature:
        call    WriteTemperature
NOCHANGEST:
        call    LEARN                           ; do the learn switch
        di
        cp      BRPM_TIME_OUT,RPM_TIME_OUT
        jr      z,TESTRPM
RESET:
        jp      START
TESTRPM:
        cp      BFORCE_IGNORE,FORCE_IGNORE
        jr      nz,RESET
        ei
        di
```

```
        cp      BAUTO_DELAY_HI,AUTO_DELAY_HI
        jr      nz,RESET
        cp      BAUTO_DELAY_LO,AUTO_DELAY_LO
        jr      nz,RESET
        cp      BCMD_DEB,CMD_DEB
        jr      nz,RESET
        cp      BSTATE,STATE
        jr      nz,RESET
        ei
TESTRS232:
        SRP     #TIMER_GROUP
        cp      RSSTART,#0FFH                   ; test for starting a transmission
        jp      z,SkipRS232                     ; if starting a trans skip
        cp      rscommand,#"Z"                  ;
        jp      ugt,SkipRS232                   ;
        cp      rscommand,#"0"                  ; test for in range
        jp      ult,SkipRS232                   ; if out of range skip
        cp      rs232docount,#12d               ; test for output done
        jp      nz,SkipRS232                    ; if not the skip
        cp      RSCCOUNT,#90H                   ; test for cr out
        jp      nz,CrOutSkip                    ; no
        call    CrOut
        jp      SkipRS232
CrOutSkip:
        di
        push    rs_temp_hi                      ; save the present value
        push    rs_temp_lo
        push    rscommand                       ; save the command
        sub     rscommand,#"0"                          ; setup for table

        ld      rs_temp_hi,#^hb RS232JumpTable  ; address pointer to table
        ld      rs_temp_lo,#^lb RS232JumpTable

        add     rs_temp_lo,rscommand            ; look up the jump 3x
        adc     rs_temp_hi,#00                  ;
        add     rs_temp_lo,rscommand            ; look up the jump 3x
        adc     rs_temp_hi,#00                  ;
        add     rs_temp_lo,rscommand            ; look up the jump 3x
        adc     rs_temp_hi,#00                  ;
        call    @rs_temp                        ; call this address
        cp      rscommand,#0FFH                 ; test for cleared command
        jr      nz,SaveCommand                  ;
        pop     rs_temp_lo                      ; throw away value
        jr      SaveCommandRet
SaveCommand:
        pop     rscommand                       ; reset the varables
SaveCommandRet:
        pop     rs_temp_lo                      ;
        pop     rs_temp_hi                      ;
        ei
        jp      SkipRS232                       ; done


RS232JumpTable:
        jp      GOTC0                           ; 30
        jp      GOTC1                           ; 31
```

```
        jp      GOTC2                           ; 32
        jp      GOTC3                           ; 33
        jp      GOTC4                           ; 34
        jp      GOTC5                           ; 35
        jp      GOTC6                           ; 36
        jp      GOTC7                           ; 37
        jp      GOTC8                           ; 38
        jp      GOTC9                           ; 39
        jp      GOTCNOP                         ; 3A :
        jp      GOTCNOP                         ; 3B ;
        jp      GOTCLT                          ; 3C <
        jp      GOTCNOP                         ; 3D =
        jp      GOTCGT                          ; 3E >
        jp      GOTCNOP                         ; 3F ?
        jp      GOTCNOP                         ; 40 @
        jp      GOTCA                           ; 41
        jp      GOTCB                           ; 42
        jp      GOTCC                           ; 43
        jp      GOTCD                           ; 44
        jp      GOTCE                           ; 45
        jp      GOTCF                           ; 46
        jp      GOTCG                           ; 47
        jp      GOTCH                           ; 48
        jp      GOTCI                           ; 49
        jp      GOTCJ                           ; 4A
        jp      GOTCK                           ; 4B
        jp      GOTCL                           ; 4C
        jp      GOTCM                           ; 4D
        jp      GOTCN                           ; 4E
        jp      GOTCO                           ; 4F
        jp      GOTCP                           ; 50
        jp      GOTCQ                           ; 51
        jp      GOTCR                           ; 52
        jp      GOTCS                           ; 53
        jp      GOTCT                           ; 54
        jp      GOTCU                           ; 55
        jp      GOTCV                           ; 56
        jp      GOTCW                           ; 57
        jp      GOTCX                           ; 58
        jp      GOTCY                           ; 59
        jp      GOTCZ                           ; 5A

SkipRS232:
        cp      R_DEAD_TIME,#20                 ; test for too long dead
        jp      nz,MAINLOOP                     ; if not loop
        clr     RADIOC                          ; clear the radio counter
        clr     RFLAG                           ; clear the radio flag
        jp      MAINLOOP                        ; loop forever

;************************************************************
; Temperature write
;************************************************************

WriteTemperature:
```

```
        ld      MTEMPH,MotorTempHi              ; get the motor temp
        ld      MTEMPL,MotorTempLo              ;
        ld      ADDRESS,#AddressTemperature     ; set the address
        ld      SKIPRADIO,#0FFH                 ; turn off the radio memory read
        call    WRITEMEMORY                     ; write the data
        clr     SKIPRADIO                       ; turn back on the radio
        ret

;*****************************************************************
; RS232 SUBROUTINES
;*****************************************************************

GOTCLT:                                         ; 3C <
        ld      Jog,#0AAH                       ; jog
        jp      OnePosC

GOTCGT:                                         ; 3E >
        ld      Jog,#055H                       ; jog
        jp      OnePosC


GOTCNOP:                                        ; no operation skip values
        jp      OnePosC

GOTC0:                                          ; SWITCH DATA
        ld      RS232DO,#"0"                    ; clear the data
        cp      CMD_DEB,#0FFH                   ; test for the command set
        jr      nz,CMDSWOPEN
        or      RS232DO,#00000001B              ; set the marking bit
CMDSWOPEN:
        cp      LIGHT_DEB,#0FFH                 ; test for the worklight set
        jr      nz,WLSWOPEN
        or      RS232DO,#00000010B              ; set the marking bit
WLSWOPEN:
        cp      VAC_DEB,#0FFH                   ; test fir the vacation set
        jp      nz,VACSWOPEN                    ;
        or      RS232DO,#00000100B              ; set the marking bit
        jp      VACSWOPEN



GOTC1:                                          ; SYSTEM STATE
        ld      RS232DO,#"0"                    ; start from 0
        cp      VACFLAG,#00H                    ; test the vacation flag
        jr      z,NOTINVACATION
        or      RS232DO,#001B                   ;

NOTINVACATION:
        tm      p0,#WORKLIGHT                   ; test for the light on
        jr      z,LIGHTISOFF
        or      RS232DO,#010B                   ; mark the bit
LIGHTISOFF:
        tm      AOBSF,#00000001B                ; test for aobs error
        jp      z,VACSWOPEN
```

```
        or      RS232DO,#100B                      ;
        jp      VACSWOPEN                          ;


GOTC2:
        ld      RS232DO,RPM_PERIOD_LO              ;
        cp      RSCCOUNT,#01H                      ; test for on transmitted last cycle
        jp      z,LastPos
        ld      RS232DO,RPM_PERIOD_HI              ;
        jp      FirstPos

GOTC3:                                             ; SWITCH DATA
        ld      RS232DO,#"0"                       ; clear the data
        cp      LEARNDB,#0FFH                      ; test for learn set
        jr      nz,LearnSwitchOpen                 ; if open skip bit
        or      RS232DO,#00000001B                 ; set the marking bit
LearnSwitchOpen:
        cp      LEARNT,#0FFH                       ; test for learn mode
        jr      z,RSNOTINLEARN
        or      RS232DO,#00000010B                 ;
RSNOTINLEARN:
        cp      WIN_FLAG,#00                       ; test for the win flag
        jp      z,VACSWOPEN                        ; if not set leave bit as 0
        or      RS232DO,#00000100B                 ;
        jp      VACSWOPEN


GOTC4:
        ld      RS232PAGE,#00H                     ;
        jp      RS232PAGEOUT

GOTC5:
        ld      RS232PAGE,#10H                     ;
        jp      RS232PAGEOUT

GOTC6:
        ld      RS232PAGE,#20H                     ;
        jp      RS232PAGEOUT


GOTC7:
        ld      RS232PAGE,#30H                     ;
        jp      RS232PAGEOUT



GOTC9:
        call    LearnSet
        jp      OnePosN


GOTCA:
        ld      rs232do,POSITION_LO                ;
        cp      RSCCOUNT,#01H                      ; test for on transmitted last cycle
        jp      z,LastPos
```

```
        ld      rs232do,POSITION_HI         ;
        jp      FirstPos

GOTCB:
        ld      rs232do,DN_LIM_LO           ;
        cp      RSCCOUNT,#01H               ; test for on transmitted last cycle
        jp      z,LastPos
        ld      RS232DO,DN_LIM_HI           ;
        jp      FirstPos

GOTCC:
        ld      rs232do,UP_LIM_LO           ;
        cp      RSCCOUNT,#01H               ; test for on transmitted last cycle
        jp      z,LastPos
        ld      rs232do,UP_LIM_HI           ;
        jp      FirstPos

GOTCD:
        ld      rs232do,MAX_F_LO            ;
        cp      RSCCOUNT,#01H               ; test for on transmitted last cycle
        jp      z,LastPos
        ld      rs232do,MAX_F_HI            ;
        jp      FirstPos

GOTCE:
        ld      rs232do,DN_FORCE_LO         ;
        cp      RSCCOUNT,#01H               ; test for on transmitted last cycle
        jp      z,LastPos
        ld      rs232do,DN_FORCE_HI
        jp      FirstPos

GOTCF:
        ld      rs232do,UP_FORCE_LO         ;
        cp      RSCCOUNT,#01H               ; test for on transmitted last cycle
        jp      z,LastPos
        ld      rs232do,UP_FORCE_HI         ;
        jp      FirstPos

GOTCG:
        ld      RS232DO,PWINDOW             ; read the state
        jp      LastPos

GOTCH:
        ld      RS232DO,WIN_FLAG            ; read the state
        add     RS232DO,#"0"
        jp      LastPos

GOTCI:                                      ; give the system a command
        ld      LAST_CMD,#0AAH
        call    CmdSet                      ; set the command
        ld      RS232ODELAY,#100D           ; set a delay of 100*.2ms = 20mS
        jp      OnePosN

GOTCJ:
        ld      RS232DO.Temperature         ; read the temperature
        jp      LastPos
```

```
GOTCK:
        ld      RS232DO,MotorTempHi             ; read the motor temperature
        jp      LastPos

GOTCL:
        cp      L_A_C,#41h                      ; test for the learn limits flag
        jr      ugt,InLearnOutForces            ; if in learn then output forces
        ld      rs232do,#"9"                    ; else 9
        jp      LastPos                         ; output
InLearnOutForces:
        ld      rs232do,FORCES                  ; output forces
        add     rs232do,#030h                   ;
        jp      LastPos


GOTCM:                                          ; give the system vacation switch action
        call    VacSet                          ; set the vacation
        jp      OnePosN

GOTCN:                                          ; give the system a work light command
        call    LightSet                        ; set the worklight switch
        jp      OnePosN

GOTCO:
        ld      rs232do,ForceAddLo              ;
        cp      RSCCOUNT,#01H                   ; test for on transmitted last cycle
        jp      z,LastPos
        ld      rs232do,ForceAddHi              ;
        jp      FirstPos

GOTCP:
        di
        ld      CMD_DEB,#00
        ld      BCMD_DEB,CMD_DEB
        jp      OnePosN

GOTCQ:
        di
        ld      CMD_DEB,#0FFH
        ld      BCMD_DEB,CMD_DEB
        jp      OnePosN

GOTCR:
        cp      RsRto,#101D                     ; test for the timer time out
        jr      ule,OutputCode                  ; if timer active then output radio code
        ld      RS232DO,#0FFH                   ;
        jp      RCodeOut
OutputCode:
        cp      RSCCOUNT,#0D                    ; test for the force byte
        jr      z,CodeRFirst
        cp      RSCCOUNT,#1D
        jr      z,CodeRSec
        cp      RSCCOUNT,#2D
        jr      z,CodeRTh
        ld      RS232DO,PRADIO1L                ;
```

```
RCodeOut:
        cp      RSCCOUNT,#3D            ; test for the end
        jp      z,LastPos
        jp      FirstPos

CodeRFirst:
        ld      RS232DO,PRADIO3H        ;
        jr      RCodeOut

CodeRSec:
        ld      RS232DO,PRADIO3L        ;
        jr      RCodeOut

CodeRTh:
        ld      RS232DO,PRADIO1H        ;
        jr      RCodeOut

GOTCS:
        cp      RSCCOUNT,#0D            ; test for the force byte
        jr      z,CodeSFirst
        cp      RSCCOUNT,#1D
        jr      z,CodeSSec
        jr      CodeSTh
SCodeOut:
        cp      RSCCOUNT,#2D            ; test for the end
        jp      z,LastPos
        jp      FirstPos

CodeSFirst:
        ld      RS232DO,#"0"            ;
        cp      Temperature,#100D       ;
        jr      ult,SCodeOut
        ld      RS232DO,#"1"            ;
        jr      SCodeOut

CodeSSec:
        push    Temperature             ; save the temperature
        cp      Temperature,#100d       ; remove the last digit
        jr      ult,SkipSSub            ;
        sub     Temperature,#100d       ;
SkipSSub:
        clr     RS232DO                 ; start at zero for the start bit
SSecLoop:
        cp      Temperature,#10d        ; test for loop continue
        jr      ult,SSecDone            ; test for done
        sub     Temperature,#10d        ;
        inc     RS232DO                 ; counter increase
        jr      SSecLoop
SSecDone:
        pop     Temperature             ; reset
        add     RS232DO,#"0"
        jr      SCodeOut                ; done


CodeSTh:
```

```
        push    Temperature                     ; save the temperature
        cp      Temperature,#100d               ; remove the last digit
        jr      ult,SkipSSub2                   ;
        sub     Temperature,#100d               ;
SkipSSub2:
        clr     RS232DO                         ; start at zero for the start bit
SThLoop:
        cp      Temperature,#10d                ; test for loop continue
        jr      ult,SThDone                     ; test for done
        sub     Temperature,#10d                ;
        inc     RS232DO                         ; counter increase
        jr      SThLoop
SThDone:
        ld      RS232DO,Temperature             ; output remainer
        pop     Temperature                     ; reset
        add     RS232DO,#"0"
        jr      SCodeOut                        ; done


GOTCT:
        cp      RSCCOUNT,#0D                    ; test for the force byte
        jr      z,CodeTFirst
        cp      RSCCOUNT,#1D
        jr      z,CodeTSec
        jr      CodeTTh
TCodeOut:
        cp      RSCCOUNT,#2D                    ; test for the end
        jp      z,LastPos
        jp      FirstPos

CodeTFirst:
        ld      RS232DO,#"0"                    ;
        cp      MotorTempHi,#100D               ;
        jr      ult,TCodeOut
        ld      RS232DO,#"1"                    ;
        jr      TCodeOut

CodeTSec:
        push    MotorTempHi                     ; save the temperature
        cp      MotorTempHi,#100d               ; remove the last digit
        jr      ult,SkipTSub                    ;
        sub     MotorTempHi,#100d               ;
SkipTSub:
        clr     RS232DO                         ; start at zero for the start bit
TSecLoop:
        cp      MotorTempHi,#10d                ; test for loop continue
        jr      ult,TSecDone                    ; test for done
        sub     MotorTempHi,#10d                ;
        inc     RS232DO                         ; counter increase
        jr      TSecLoop
TSecDone:
        pop     MotorTempHi                     ; reset
        add     RS232DO,#"0"
        jr      TCodeOut                        ; done
```

```
CodeTTh:
        push    MotorTempHi                 ; save the temperature
        cp      MotorTempHi,#100d           ; remove the last digit
        jr      ult,SkipTSub2               ;
        sub     MotorTempHi,#100d           ;
SkipTSub2:
        clr     RS232DO                     ; start at zero for the start bit
TThLoop:
        cp      MotorTempHi,#10d            ; test for loop continue
        jr      ult,TThDone                 ; test for done
        sub     MotorTempHi,#10d            ;
        inc     RS232DO                     ; counter increase
        jr      TThLoop
TThDone:
        ld      RS232DO,MotorTempHi         ; output remainer
        pop     MotorTempHi                 ; reset
        add     RS232DO,#"0"
        jr      TCodeOut                    ; done


GOTCU:
        ld      RsMode,#232D                ; turn on the rs232 mode period
        ld      RS232DO,#Version            ; read the Version
        and     rs232do,#00001111B          ; get the last byte
        add     rs232do,#"0"                ;
        cp      RSCCOUNT,#01H               ; test for on transmitted last cycle
        jp      z,LastPos
        ld      rs232do,#Version            ; read the Version
        swap    rs232do
        and     rs232do,#00001111B          ; get the first byte
        add     rs232do,#"0"                ;
        jp      FirstPos

GOTCV:
        ld      RS232DO,STATE               ; read the state
        add     RS232DO,#"0"                ; add the offset
        jp      VACSWOPEN                   ;

GOTCW:
        ld      RS232DO,STACKREASON         ; read the reason
        swap    RS232DO                     ;
        add     RS232DO,#"0"                ; add the offset
        jp      VACSWOPEN                   ;

GOTCX:
        ld      RS232DO,FAULTCODE           ; read the fault
        add     RS232DO,#"0"                ; add the offset
        jp      VACSWOPEN                   ;

GOTCY:
        clr     RS232DO                     ; start clean
        tm      P0,#00010000B               ; test for first gear strap
        jr      z,SkipStrap1
        or      RS232DO,#00000001b          ; set the bit
```

```
SkipStrap1:
        tm      P0,#00100000B                   ; test for the second gear
        jr      z,SkipStrap2                    ;
        or      RS232DO,#00000010B              ; set the bit
SkipStrap2:
        tm      P2,#10000000B                   ; test for the temperature strap
        jr      z,SkipStrap3                    ;
        or      RS232DO,#00000100B              ; set the bit
SkipStrap3:
        add     RS232DO,#"0"                    ; add the offset
        jp      VACSWOPEN                       ;


GOTCZ:
        ld      MotorTempHi,Temperature
        call    WriteTemperature
        jp      OnePosN


;*************************************************************
; Store the limits and the up force settings
;*************************************************************

STOREFL:
        ld      SKIPRADIO,#0FFH                 ;
        ld      ADDRESS,#AddressUpLimit         ; set non vol address to the up limit
        ld      MTEMPH,UP_LIM_HI                ; save into nonvolital
        ld      MTEMPL,UP_LIM_LO                ;
        call    WRITEMEMORY                     ; write the value

        ld      ADDRESS,#AddressDownLimit       ; set non vol address to the down limit
        ld      MTEMPH,DN_LIM_HI                ; save into nonvolital
        ld      MTEMPL,DN_LIM_LO                ;
        call    WRITEMEMORY                     ; write the value

StoreUpForceTable:
        ld      ADDRESS,#AddressUpForceTable
        call    StoreForceTable
        inc     WIN_FLAG                        ; turn on the window
        clr     SKIPRADIO                       ;
        JP      CLRLAC                          ; return and clear the lac


FirstPos:
        dec     RSSTART                         ; set the start flag
        inc     RSCCOUNT                        ; increase the count
        ret
OnePosN:
        ld      RS232DO,#"0"                    ;
        jr      LastPos
OnePosC:
        ld      RS232DO,#"@"                    ;
```

```
LastPos:
VACSWOPEN:
        ld      RSCCOUNT,#090H          ; mark to do cr
        dec     RSSTART                 ; set the start flag
        ret
CrOut:
        ld      RS232DO,#00DH           ; set the cr output
        clr     RSCCOUNT                ; reset the counter
        dec     RSSTART                 ; set the start flag
        ld      rscommand,#0FFH         ; turn off command
        ret

RS232PAGEOUT:
        ld      SKIPRADIO,#0FFH         ; set the skip radio flag
        ld      ADDRESS,RSCCOUNT        ; find the address
        rcf                             ;
        rrc     ADDRESS                 ;
        or      ADDRESS,RS232PAGE       ;
        call    READMEMORY              ; read the data
        ld      RS232DO,MTEMPH
        tm      RSCCOUNT,#01H           ; test which byte
        jr      z,RPBYTE
        ld      RS232DO,MTEMPL
RPBYTE:
        clr     SKIPRADIO               ; turn off the skip radio
        cp      RSCCOUNT,#1FH           ; test for the end
        jr      z,LastPos
        jr      FirstPos


GOTC8:
        ld      RS232DO,#0FFH           ; flag set to error to start
        ld      SKIPRADIO,#0FFH         ; set the skip radio flag
        ld      MTEMPH,#0FFH            ; set the data to write
        ld      MTEMPL,#0FFH            ;
        ld      ADDRESS,#00             ; start at address 00
WRITELOOP1:
        .IF     E21
        xor     P1,#00000001B           ; Kick the external dog
        .ELSE
        WDT                             ; KICK THE DOG
        .ENDIF
        call    WRITEMEMORY             ;
        inc     ADDRESS                 ; do the next address
        cp      ADDRESS,#40H            ; test for the last address
        jr      nz,WRITELOOP1
        ld      ADDRESS,#00             ; start at address 0
READLOOP1:
        .IF     E21
        xor     P1,#00000001B           ; Kick the external dog
        .ELSE
        WDT                             ; KICK THE DOG
        .ENDIF
        call    READMEMORY              ; read the data
        inc     MTEMPH                  ; test the high
        jr      nz,MEMORYERROR          ; if error mark
```

```
        inc     MTEMPL                          ; test the low
        jr      nz,MEMORYERROR                  ; if error mark
        inc     ADDRESS                         ; set the next address
        cp      ADDRESS,#40H                    ; test for the last address
        jr      nz,READLOOP1

        ld      MTEMPH,#000H                    ; set the data to write
        ld      MTEMPL,#000H                    ;
        ld      ADDRESS,#00                     ; start at address 00
WRITELOOP2:
        .IF     E21
        xor     P1,#00000001B                   ; Kick the external dog
        .ELSE
        WDT                                     ; KICK THE DOG
        .ENDIF
        call    WRITEMEMORY                     ;
        inc     ADDRESS                         ; do the next address
        cp      ADDRESS,#40H                    ; test for the last address
        jr      nz,WRITELOOP2
        ld      ADDRESS,#00                     ; start at address 0
READLOOP2:
        .IF     E21
        xor     P1,#00000001B                   ; Kick the external dog
        .ELSE
        WDT                                     ; KICK THE DOG
        .ENDIF
        call    READMEMORY                      ; read the data
        cp      MTEMPH,#00                      ; test the high
        jr      nz,MEMORYERROR                  ; if error mark
        cp      MTEMPL,#00                      ; test the low
        jr      nz,MEMORYERROR                  ; if error mark
        inc     ADDRESS                         ; set the next address
        cp      ADDRESS,#40H                    ; test for the last address
        jr      nz,READLOOP2
        call    CLEARCODES
        clr     SKIPRADIO                       ; clear the skip radio flag
        clr     RS232DO                         ; flag all ok
MEMORYERROR:
        dec     RSSTART                         ; set the start flag
        ld      RSCOMMAND,#0FFH                 ; turn off command
        jp      SkipRS232                       ; return


;**************************************************************
; PORT INITILIZATION
;**************************************************************
PORTINIT:
        ld      P0,#P01S_INIT                   ; RESET all ports
        ld      P2,#P2S_INIT                    ;
        ld      P3,#P3S_INIT                    ;
PORTREF:                                        ; port refresh
        ld      P01M,#P01M_INIT                 ; set mode p00-p03 out p04-p07in
        ld      P3M,#P3M_INIT                   ; set port3 p30-p33 input analog mode
                                                ; p34-p37 outputs
        ld      P2M,#(P2M_INIT+0)               ; set port 2 mode
```

```
        ret                                         ; return

;*************************************************************
; Radio interrupt from a edge of the radio signal
;*************************************************************

RADIO_INT:
        push    RP                      ; save the radio pair
        srp     #RADIO_GRP              ; set the register pointer '
        ld      rtemph.T0EXT            ; read the upper byte
        ld      rtempl.T0               ; read the lower byte
        tm      IRQ.#00010000B          ; test for pending int
        jr      z,RTIMEOK               ; if not then ok time
        tm      rtempl.#10000000B       ; test for timer reload
        jr      z,RTIMEOK               ; if not reloaded then ok
        dec     rtemph                  ; if reloaded then dec high for sync
RTIMEOK:
        clr     R_DEAD_TIME             ; clear the dead time
        .IF E21
        and     IMR,#11111100B          ; turn off the radio interrupt
        .ELSE
        and     IMR,#11111110B          ; turn off the radio interrupt
        .ENDIF
        ld      rtimedh,rtimeph         ; find the difference
        ld      rtimedl,rtimepl         ;
        sub     rtimedl,rtempl          ;
        sbc     rtimedh,rtemph          ;  past time and the past time in temp
        tm      rtimedh.#10000000B      ; test for a negitive number
        jr      z.RTIMEDONE             ; if the number is not negitive then done
        ld      rtimedh,rtemph          ; find the difference
        ld      rtimedl.rtempl          ;
        sub     rtimedl,rtimepl         ;
        sbc     rtimedh,rtimeph         ;past time and the past time in temp
RTIMEDONE:
        tm      P3,#00000100B           ; test the port for the edge
        jr      nz.ACTIVETIME           ; if it was the active time then branch
INACTIVETIME:
        cp      RINFILTER,#0FFH         ; test for active last time
        jr      z,GOINACTIVE            ; if so continue
        jr      RADIO_EXIT              ; if not the return
GOINACTIVE:
        .IF E21
        .ELSE
        or      IRQ,#01000000B          ; set the bit setting direction to pos edge
        .ENDIF
        clr     RINFILTER               ; set flag to inactive
        ld      rtimeih,rtimedh         ; transfer difference to inactive
        ld      rtimeil,rtimedl         ;
        ld      rtimeph,rtemph          ; transfer temp into the past
        ld      rtimepl,rtempl          ;
        jr      RADIO_EXIT              ; return
ACTIVETIME:
        cp      RINFILTER,#00H          ; test for active last time
```

```
        jr      z,GOACTIVE              ; if so continue
        jr      RADIO_EXIT              ; if not the return
GOACTIVE:
        .IF E21
        .ELSE
        and     IRQ,#00111111B          ; clear the bit setting dir to neg edge
        .ENDIF
        ld      RINFILTER,#0FFH         ;
        ld      rtimeah,rtimedh         ; transfer difference to active
        ld      rtimeal,rtimedl         ;
        ld      rtimeph,rtemph          ; transfer temp into the past
        ld      rtimepl,rtempl          ;
        ei
        cp      radioc,#00H             ; test for blank time
        jr      nz,INSIGNAL             ; if the count not zero then in signal
MEASUREBLANK:
        cp      rtimeih,#110D           ; test the timer for > 55mS
        jp      ugt,CLEARRADIO          ; if > 55 then clear the radio
        cp      rtimeih,#40D            ; test the timer for < 20mS
        jp      ult,CLEARRADIO          ; if < 20mS then clear the radio
        cp      rtimeah,#03H            ; test the sync for a 3mS period first > 1
        jr      ugt,SETREC3MS           ; if 2mS or greater then 3mS sync code
        jr      nz,SETREC1MS            ; if less then 1 then it is a 1mS
        cp      rtimeal,#09DH           ; test for 1.85 "middle value 2"
        jr      ugt,SETREC3MS           ; if greater then set a 3
SETREC1MS:
        tm      RFLAG,#00010000B        ; test for the reception of the 1mS code
        jr      z,SETFIRST1MS           ; if the bit is not set then this is the first
1ms
        and     RFLAG,#10111111B        ; clear the flag so writing into 3mS word
        or      RFLAG,#00100000B        ; set the flag saying 2nd 1mS word
        clr     radio3h                 ; clear the last reception
        clr     radio3l                 ;
        jr      INCCOUNT                ; then inc the count for insignal
SETFIRST1MS:
        or      RFLAG,#01000000B        ; set the flag for the first 1mS word
        clr     radio1h                 ; clear the last reception
        clr     radio1l                 ;
        jr      INCCOUNT                ; then inc the count for insignal
SETREC3MS:
        and     RFLAG,#10111111B        ; clear the flag so writing into 3mS word
        clr     radio3h                 ; c'ear the last reception
        clr     radio3l                 ;
INCCOUNT:
        inc     radioc                  ; set the counter to the next word
        jr      RADIO_EXIT

RADIO_EXIT:
        pop     RP                      ; reset the register pair
        iret

INSIGNAL:
        cp      rtimeah,#9D             ; test the radio pulse width for 4.5mS
        jp      ugt,CLEARRADIO          ; if greater then 4.5 then clear the radio
PULSEWOK:
        cp      rtimeih,#9D             ; test the radio blank width for 4.5mS
```

```
        jp      ugt,CLEARRADIO          ; if greater then 4.5 then clear the radio
BLANKWOK:
        ld      rtemph,rtimeih          ; transfer pulse time to temp reg
        ld      rtempl,rtimeil          ;
        sub     rtempl,rtimeal          ; subtract the pulse from the blank
        sbc     rtemph,rtimeah          ;
        jr      c,NEGDIFF               ; if the difference is negitive then branch
        cp      rtemph,#01H             ; test for a number 1
        jr      ugt,SETTO0              ; if greater then set 0
        jr      ult,SETTO1              ; if less then 1 set to 1
        tm      rtempl,#10000000B       ; test for 80 or greater
        jr      z,SETTO1                ; if the diff is less then 80h
        jr      SETTO0                  ; else set to a zero
NEGDIFF:
        ld      rtemph,rtimeah          ; transfer pulse time to temp reg
        ld      rtempl,rtimeal          ;
        sub     rtempl,rtimeil          ; subtract the pulse from the blank
        sbc     rtemph,rtimeih          ;
        cp      rtemph,#01H             ; test for a number 1
        jr      ugt,SETTO2              ; if greater then set 2
        jr      ult,SETTO1              ; if less then 1 set to 1
        tm      rtempl,#10000000B       ; test for 80 or greater
        jr      z,SETTO1                ; if the diff is less then 80h one
        jr      SETTO2                  ; else set to a two
SETTO0:
        ld      RTEMP,#00D              ; set the bit value to a 00
        jr      INCRECORD               ; goto adding into the record
SETTO1:
        ld      RTEMP,#01D              ; set the bit value to a 01
        jr      INCRECORD               ; goto adding into the record
SETTO2:
        ld      RTEMP,#02D              ; set the bit value to a 10
        jr      INCRECORD               ; goto adding into the record

INCRECORD:
        tm      RFLAG,#01000000B        ; test radio flag for area to be modifing
        jr      z,MS3RECORD             ;  if cleared then working the 3ms
        ld      rtemph,radio1h          ; transfer the record to temp
        ld      rtempl,radio1l          ;
        add     radio1l,rtempl          ; add the number to it self 2* for base 3
        adc     radio1h,rtemph          ;
        add     radio1l,rtempl          ;
        adc     radio1h,rtemph          ;
        add     radio1l,rtemp           ;
        adc     radio1h,#00h            ;
        inc     radioc                  ; increase the radio counter
        cp      radioc,#11D             ; test for the last bit
        jr      z,GOTAWORD              ; if so we got a word
        jp      ugt,CLEARRADIO          ; else garbage
        jr      RADIO_EXIT              ; else return till the next bit comes along

MS3RECORD:
        ld      rtemph,radio3h          ; transfer the record to temp
        ld      rtempl,radio3l          ;
        add     radio3l,rtempl          ; add the number to it self 2* for base 3
        adc     radio3h,rtemph          ;
```

```
        add     radio3l,rtempl                  ;
        adc     radio3h,rtemph                  ;
        add     radio3l,rtemp                   ; add in the new value
        adc     radio3h,#00D                    ;
        inc     radioc                          ; increase the radio counter
        cp      radioc,#11D                     ; test for the last bit
        jr      z,GOTAWORD                      ; if so we got a word
        jp      RADIO_EXIT                      ; else return till the next bit comes along

GOTAWORD:
        tm      RFLAG,#01000000B                ; test radio flag for area just modifing
        jr      z,MARK3REC                      ;  if bit is cleared then the 3ms is filled
        or      RFLAG,#00010000B                ; set the flag
        jr      TESTFORTWO                      ; jump to test for two codes
MARK3REC:
        or      RFLAG,#00001000B                ; set the flag
        jr      TESTFORTWO                      ; jump to test for two codes
DONEONE:
        clr     radioc                          ; clear the radio counter
        jp      RADIO_EXIT                      ; return
TESTFORTWO:
        tm      RFLAG,#00010000B                ; test for the 1mS word
        jr      z,DONEONE                       ; we just have one code done
        tm      RFLAG,#00001000B                ; test for the 3mS word
        jr      z,DONEONE                       ; we just have one code done
        tm      RFLAG,#00100000B                ; test the flag for BC
        jr      z,KNOWCODE                      ; if A code we do nothing
        or      RFLAG,#00000010B                ; set the B and C flag
        cp      rtemp,#00                       ; test word 10 for a 0 "C" code
        jp      z,KNOWCODE                      ; if a C code were done
        or      RFLAG,#00000100B                ; set the B code flag
KNOWCODE:
        clr     RsRto                           ; reset the received flag

        cp      SKIPRADIO,#0FFH                 ; test for the skip flag
        jp      z,CLEARRADIO                    ; skip flag active donot look at EE mem

        ld      ADDRESS,#AddressVacation        ; set the non vol to the VAC flag
        call    READMEMORY                      ; read the value
        ld      VACFLAG,MTEMPH                  ; save into volital
        cp      LEARNT,#0FFH                    ; test for in learn mode
        jr      z,TESTCODE                      ; if out of learn mode then test matching
STORECODE:
        cp      PRADIO1H,radio1h                ; test for the match
        jr      nz,STORENOTMATCH                ; if not a match then loop again
        cp      PRADIO1L,radio1l                ; test for the match
        jr      nz,STORENOTMATCH                ; if not a match then loop again
        cp      PRADIO3H,radio3h                ; test for the match
        jr      nz,STORENOTMATCH                ; if not a match then loop again
        cp      PRADIO3L,radio3l                ; test for the match
        jr      nz,STORENOTMATCH                ; if not a match then loop again
        call    TESTCODES                       ; test the code to see if in memory now
        cp      ADDRESS,#0FFH                   ;
        jr      nz,NOWRITESTORE                 ; if there is a match pretend to store

STOREMATCH:
```

```
        tm      RFLAG,#00000100B            ; test for the b code
        jr      nz,BCODE                    ; if a B code jump
        tm      RFLAG,#00000010B            ; test for a C code
        jr      nz,CCODE                    ; if a C code jump
ACODE:
        ld      ADDRESS,#AddressApointer    ; set the address to read the last written
        call    READMEMORY                  ; read the memory
        inc     MTEMPH                      ; add 2 to the last written
        inc     MTEMPH                      ;
        and     MTEMPH,#11111110B           ; set the address on a even number
        cp      MTEMPH,#17H                 ; test for the last address
        jr      ult,GOTAADDRESS             ; if not the last address jump
        ld      MTEMPH,#00D                 ; set the address to 0
GOTAADDRESS:
        ld      ADDRESS,#AddressApointer    ; set the address to write the last written
        ld      RTEMP,MTEMPH                ; save the address
        ld      MTEMPL,MTEMPH               ; both bytes same
        call    WRITEMEMORY                 ; write it
        ld      ADDRESS,rtemp               ; set the address
        jr      READYTOWRITE                ;
BCODE:
        ld      ADDRESS,#AddressB           ; set the address for the B code
        jr      READYTOWRITE                ;
CCODE:
        ld      ADDRESS,#AddressC           ; set the address for the C code
READYTOWRITE:
        call    WRITECODE                   ; write the code in radio1 and radio3
NOWRITESTORE:
        xor     p0,#WORKLIGHT               ; toggle light
        ld      LearnLed,#00111111b         ; turn off the LED for program mode
        ld      LIGHT1S,#244D               ; turn on the 1 second blink
        ld      LEARNT,#0FFH                ; set learnmode timer
        clr     RTO                         ; disallow cmd from learn
        jp      CLEARRADIO                  ; return
STORENOTMATCH:
        ld      PRADIO1H,radio1h            ; transfer radio into past
        ld      PRADIO1L,radio1l            ;
        ld      PRADIO3H,radio3h            ;
        ld      PRADIO3L,radio3l            ;
        jp      CLEARRADIO                  ; get the next code


TESTCODE:
        ld      PRADIO1H,radio1h            ; transfer radio into past
        ld      PRADIO1L,radio1l            ;
        ld      PRADIO3H,radio3h            ;
        ld      PRADIO3L,radio3l            ;
        tm      LearnLed,#11000000B         ; test for fault or learn
        jr      nz,FS1                      ; if so then skip blink
        ld      LearnLed,#00111100b         ; blink led
FS1:
        call    TESTCODES                   ; test the codes
        cp      ADDRESS,#0FFH               ; test for the not matching state
        jr      nz,GOTMATCH                 ; if matching send a command if needed
        jp      CLEARRADIO                  ; else clear the radio
```

```
GOTMATCH:
        or      RFLAG,#00000001B        ; set the flag for recieving without error
        cp      RTO,#101D               ; test for the timer time out
        jr      ult,NOTNEWMATCH         ; if timer active then donot reissue cmd
TESTVAC:
        cp      VACFLAG,#00B            ; test for the vacation mode
        jr      z,TSTSDISABLE           ; if not vac mode disable

        cp      ADDRESS,#AddressB+1     ; test for the B code
        jr      nz,NOTNEWMATCH          ; if not a B not a match
TSTSDISABLE:
        cp      SDISABLE,#32D           ; test for 4 second
        jr      ult,NOTNEWMATCH         ; if 6 s not up not a new code
        clr     RTO                     ; clear the radio timeout
        cp      ONEP2,#00               ; test for the 1.2 second time out
        jr      nz,NOTNEWMATCH          ; if timer is active then skip command
RADIOCOMMAND:
        clr     RTO                     ; clear the radio timeout
        cp      ADDRESS,#AddressB+1     ; test for a B code
        jr      nz,BDONTSET             ; if not a b code donot set flag
        ld      BCODEFLAG,#077H         ; flag for aobs bypass
BDONTSET:

        clr     LAST_CMD                ; mark the last command as radio
        ld      RADIO_CMD,#0AAH         ; set the radio command
        jr      CLEARRADIO              ; return


TESTCODES:
        ei
        clr     ADDRESS                 ; start address is 0
NEXTCODE:
        call    READMEMORY              ; read the word at this address
        cp      MTEMPH,radio1h          ; test for the match
        jr      nz,NOMATCH              ; if not matching then do next address
        cp      MTEMPL,radio1l          ; test for the match
        jr      nz,NOMATCH              ; if not matching then do next address
        inc     ADDRESS                 ; set the second half of the code
        call    READMEMORY              ; read the word at this address
        cp      MTEMPH,radio3h          ; test for the match
        jr      nz,NOMATCH2             ; not matching then do the next address
        cp      MTEMPL,radio3l          ; test for the match
        jr      nz,NOMATCH2             ; if not matching do the next address
        ret                             ; return with the address of the match

NOMATCH:
        inc     ADDRESS                 ; set the address to the next code
NOMATCH2:
        inc     ADDRESS                 ; set the address to the next code
        cp      ADDRESS,#AddressCounter ; test for the last address
        jr      ult,NEXTCODE            ; if not the last address then try again

GOTNOMATCH:
        ld      ADDRESS,#0FFH           ; set the no match flag
        ret                             ; and return
```

```
NOTNEWMATCH:
        clr     RTO                             ; reset the radio time out
        and     RFLAG,#00000001B                ; clear radio flags recieving w/o error
        clr     radioc                          ; clear the radio bit counter
        ld      LEARNT,#0FFH                    ; set learn timer "turn off" and backup
        jp      RADIO_EXIT                      ; return

CLEARRADIO:
        .IF E21
        .ELSE
        and     IRQ,#00111111B                  ; clear bit setting direction to neg edge
        .ENDIF
        ld      RINFILTER,#0FFH                 ; set flag to active
CLEARRADIOA:
        tm      RFLAG,#00000001B                ; test for receiving without error
        jr      z,SKIPRTO                       ; if flag not set then donot clear timer
        clr     RTO                             ; clear radio timer
SKIPRTO:
        clr     radioc                          ; clear the radio counter
        clr     RFLAG                           ; clear the radio flag
        jp      RADIO_EXIT                      ; return


;*******************************************************************
;
;
;       Store the force table
;       Enter with the address pointing to the first address
;
;*******************************************************************
StoreForceTable:
        push    RP                              ; set the rp
        srp     #ForceTable2                    ;
        di
        .IF     E21
        xor     P1,#00000001B                   ; Kick the external dog
        .ELSE
        WDT                                     ; KICK THE DOG
        .ENDIF
        ld      forcetemp,#14d                  ; set the number to do
        ld      forceaddress,#Force0Hi          ; set the start address
MemTransfer:
        ld      MTEMPH,@forceaddress            ; get the value
        inc     forceaddress                    ;
        ld      MTEMPL,@forceaddress
        inc     forceaddress                    ;
        .IF     E21
        xor     P1,#00000001B                   ; Kick the external dog
        .ELSE
        WDT                                     ; KICK THE DOG
        .ENDIF
        call    WRITEMEMORY                     ; write the values
        inc     ADDRESS                         ; set to the next address
        djnz    forcetemp,MemTransfer           ; loop till done
        pop     RP
```

```
        ei
        ret

;*****************************************************************
;
;       Read Force Table
;       Enter with the address pointing to the first address
;
;*****************************************************************
;
ReadForceTable:
        push    RP                              ; set the rp
        srp     #ForceTable2                    ;
        ld      SKIPRADIO,#0FFH                 ; turn off the radio
        .IF     E21
        xor     P1,#00000001B                   ; Kick the external dog
        .ELSE
        WDT                                     ; KICK THE DOG
        .ENDIF
        ld      forcetemp,#14d                  ; set the number to do
        ld      forceaddress,#Force0Hi          ; set the start address
ReadMemTransfer:
        call    READMEMORY                      ; read the value

        ld      @forceaddress,MTEMPH            ; get the value
        inc     forceaddress                    ;
        ld      @forceaddress,MTEMPL
        inc     forceaddress                    ;
        .IF     E21
        xor     P1,#00000001B                   ; Kick the external dog
        .ELSE
        WDT                                     ; KICK THE DOG
        .ENDIF
        inc     ADDRESS                         ; set to the next address
        djnz    forcetemp,ReadMemTransfer       ; loop till done
        pop     RP
        jp      ReadLimits

;*****************************************************************
;
; TIMES OUT THE LEARN MODE 30 SECONDS
; DEBOUNCES THE LEARN SWITCH FOR ERASE 6 SECONDS
;*****************************************************************
;
LEARN:

        cp      LEARNDB,#0E0H                   ; test for in learn mode
        jr      uge,LearnStillSet               ; if set test erase timer
        clr     ERASET                          ; else clear the timer
        jr      EraseTestDone                   ;
LearnStillSet:
        cp      ERASET,#48d                     ; test for the 6 seconds
        jr      nz,EraseTestDone                ; if not 6 sec keep testing
        inc     ERASET                          ; one shot
        ld      LearnLed,#00111111b             ; turn off the led
        ld      LEARNT,#0FFH                    ; set the learn timer
        ld      SKIPRADIO,#0FFH                 ; turn off the radio
        call    CLEARCODES                      ; clear the radio codes
        clr     SKIPRADIO                       ; turn back on the radio
```

```
EraseTestDone:
        cp      LEARNT,#240d                    ; test for 30 seconds timeout
        jr      z,TurnOffLearn                  ; if so turn off learn
        ret
TurnOffLearn:
        ld      LearnLed,#00111111b             ; turn off the led
        ld      LEARNT,#0FFH                    ; set the learn timer
        ret


;**************************************************************
;
; WRITE WORD TO MEMORY
; ADDRESS IS SET IN REG ADDRESS
; DATA IS IN REG MTEMPH AND MTEMPL
; RETURN ADDRESS IS UNCHANGED
;**************************************************************
;
WRITEMEMORY:
        push    RP                              ; SAVE THE RP
        srp     #LEARNEE_GRP                    ; set the register pointer

        call    STARTB                          ; output the start bit
        ld      serial,#00110000B               ; set byte to enable write
        call    SERIALOUT                       ; output the byte
        and     csport,#csl                     ; reset the chip select
        call    STARTB                          ; output the start bit
        ld      serial,#01000000B               ; set the byte for write
        or      serial,address                  ; or in the address
        call    SERIALOUT                       ; output the byte
        ld      serial,mtemph                   ; set the first byte to write
        call    SERIALOUT                       ; output the byte
        ld      serial,mtempl                   ; set the second byte to write
        call    SERIALOUT                       ; output the byte
        call    ENDWRITE                        ; wait for the ready status
        call    STARTB                          ; output the start bit
        ld      serial,#00000000B               ; set byte to disable write
        call    SERIALOUT                       ; output the byte
        and     csport,#csl                     ; reset the chip select
        pop     RP                              ; reset the RP
        ret

;**************************************************************
;
; READ WORD FROM MEMORY
; ADDRESS IS SET IN REG ADDRESS
; DATA IS RETURNED IN REG MTEMPH AND MTEMPL
; ADDRESS IS UNCHANGED
;**************************************************************
;
READMEMORY:
        push    RP                              ;
        srp     #LEARNEE_GRP                    ; set the register pointer

        call    STARTB                          ; output the start bit
        ld      serial,#10000000B               ; preamble for read
        or      serial,address                  ; or in the address
        call    SERIALOUT                       ; output the byte
        call    SERIALIN                        ; read the first byte
```

```
        ld      mtemph,serial                   ; save the value in mtemph
        call    SERIALIN                        ; read teh second byte
        ld      mtempl,serial                   ; save the value in mtempl
        and     csport,#csl                     ; reset the chip select
        pop     RP                              ;
        ret

;**************************************************************
; WRITE CODE TO 2 MEMORY ADDRESS
; CODE IS IN RADIO1H RADIO1L RADIO3H RADIO3L
;**************************************************************
WRITECODE:
        push    RP                              ;
        srp     #LEARNEE_GRP                    ; set the register pointer
        ld      mtemph,RADIO1H                  ; transfer  radio 1 to the temps
        ld      mtempl,RADIO1L                  ;
        call    WRITEMEMORY                     ; write the temp bits
        inc     address                         ; next address
        ld      mtemph,RADIO3H                  ; transfer  radio 3 to the temps
        ld      mtempl,RADIO3L                  ;
        call    WRITEMEMORY                     ; write the temps
        pop     RP                              ;
        ret                                     ; return

;**************************************************************
; CLEAR ALL RADIO CODES IN THE MEMORY
;**************************************************************

CLEARCODES:
        push    RP                              ;
        srp     #LEARNEE_GRP                    ; set the register pointer
        ld      RADIO1H,#0FFH                   ; set the codes to illegal codes
        ld      RADIO1L,#0FFH                   ;
        ld      RADIO3H,#0FFH                   ;
        ld      RADIO3L,#0FFH                   ;
        ld      address,#00H                    ; clear address 0
CLEARC:
        call    WRITECODE                       ; "A0"
        inc     address                         ; set the next address
        cp      address,#AddressCounter         ; test for the last address of radio
        jr      ult,CLEARC
        clr     mtemph                          ; clear data
        clr     mtempl
        ld      address,#AddressApointer        ; clear address F
        call    WRITEMEMORY                     ;
        pop     RP                              ;
        ret                                     ; return

;**************************************************************
; START BIT FOR SERIAL NONVOL
; ALSO SETS DATA DIRECTION AND AND CS
;**************************************************************
STARTB:
        and     csport,#csl                     ;
```

```
        and     clkport,#clockl                 ; start by clearing the bits
        and     dioport,#dol                    ;
        ld      P2M,#(P2M_INIT+0)               ; set port 2 mode output mode data
        or      csport,#csh                     ; set the chip select
        or      dioport,#doh                    ; set the data out high
        or      clkport,#clockh                 ; set the clock
        and     clkport,#clockl                 ; reset the clock low
        and     dioport,#dol                    ; set the data low
        ret                                     ; return
;*************************************************************
;
; END OF CODE WRITE
;*************************************************************
;
ENDWRITE:
        and     csport,#csl                     ; reset the chip select
        nop                                     ; delay
        or      csport,#csh                     ; set the chip select
        ld      P2M,#(P2M_INIT+4)               ; set port 2 mode input mode data
ENDWRITELOOP:
        ld      mtemp,dioport                   ; read the port
        and     mtemp,#doh                      ; mask
        jr      z,ENDWRITELOOP                  ; if bit is low then loop till we are done
        and     csport,#csl                     ; reset the chip select
        ld      P2M,#(P2M_INIT+0)               ; set port 2 mode forcing output mode
        ret

;*************************************************************
;
; SERIAL OUT
; OUTPUT THE BYTE IN SERIAL
;*************************************************************
;
SERIALOUT:
        ld      P2M,#(P2M_INIT+0)               ; set port 2 mode output mode data
        ld      mtemp,#8H                       ; set the count for eight bits
SERIALOUTLOOP:
        rlc     serial                          ; get the bit to output into the carry
        jr      nc,ZEROOUT                      ; output a zero if no carry
ONEOUT:
        or      dioport,#doh                    ; set the data out high
        or      clkport,#clockh                 ; set the clock high
        and     clkport,#clockl                 ; reset the clock low
        and     dioport,#dol                    ; reset the data out low
        djnz    mtemp,SERIALOUTLOOP
                                                ; loop till done
        ret                                     ; return
ZEROOUT:
        and     dioport,#dol                    ; reset the data out low
        or      clkport,#clockh                 ; set the clock high
        and     clkport,#clockl                 ; reset the clock low
        and     dioport,#dol                    ; reset the data out low
        djnz    mtemp,SERIALOUTLOOP
                                                ; loop till done
        ret                                     ; return

;*************************************************************
;
```

```
; SERIAL IN
; INPUTS A BYTE TO SERIAL
;*****************************************************************
;
SERIALIN:
        ld      P2M,#(P2M_INIT+4)           ; set port 2 mode input mode data
        ld      mtemp,#8H                   ; set the count for eight bits
SERIALINLOOP:
        or      clkport,#clockh             ; set the clock high
        rcf                                 ; reset the carry flag
        push    mtemp                       ; save temp
        ld      mtemp,dioport               ; read the port
        and     mtemp,#doh                  ; mask out the bits
        jr      z,DONTSET
        scf                                 ; set the carry flag
DONTSET:
        pop     mtemp                       ; reset the temp value
        rlc     serial                      ; get the bit into the byte
        and     clkport,#clockl             ; reset the clock low
        djnz    mtemp,SERIALINLOOP
                                            ; loop till done
        ret                                 ; return

;*****************************************************************
;
; TIMER UPDATE FROM INTERUPT EVERY .5mS
;*****************************************************************
;
Timer1Int:
        push    RP                          ; save the rp
        SRP     #TIMER_GROUP                ;
        dec     T0EXT
FINDTASK:
        tm      T0EXT,#00000001B            ; test for odd numbers
        jr      nz,TASK1357EXIT             ; if odd
        tm      T0EXT,#00000010B            ; test for 2 6 or 0 4
        jr      nz,TASK26                   ; if 26 then jump

TASK04:
        or      IMR,#RadioOffIMR            ; turn on the interrupt except the radio
        cp      L_A_C,#042H                 ; test for the learn force limit mode
        jr      uge,RadioOffSkip
        or      IMR,#RETURN_IMR             ; turn on the interrupt
RadioOffSkip:
        ei
        pop     rp
        iret

TASK26:
        or      IMR,#RadioOffIMR            ; turn on the interrupt except the radio
        cp      L_A_C,#042H                 ; test for the learn force limit mode
        jr      uge,Radio26OffSkip
        or      IMR,#RETURN_IMR             ; turn on the interrupt
Radio26OffSkip:
        ei
        call    STATEMACHINE                ; do the motor function
        pop     rp                          ; return the rp
        iret
```

```
TASK1357EXIT
        or      IMR,#RadioOffIMR               ; turn on the interrupt except the radio
        cp      L_A_C,#042H                    ; test for the learn force limit mode
        jr      uge,Radio1357OffSkip
        or      IMR,#RETURN_IMR                ; turn on the interrupt
Radio1357OffSkip:
        ei
        tm      T0EXT,#00000001B               ; test for state a 1 in b0
        jr      z,ONEMS
        tm      T0EXT,#00000010B               ; test for state a 1 in b1
        jr      z,ONEMS
        call    AUXLIGHT                       ;

ONEMS:
        inc     VACFLASH                       ; flash timer
;       tm      P3,#00000001B                  ; test the protector input
;       jr      z,CountActive                  ; if zero count the time
;       cp      ProtectorSwitch,#46d           ; test for the min count
;       jr      ult,ZeroProtectorCounter       ; if less the zero counter
;       cp      ProtectorSwitch,#54d           ; test for the max count
;       jr      ugt,ZeroProtectorCounter       ; if greater zero the counter
;       clr     RsTimer                        ; turn on the rs232 port
;       ld      ProtectorSwitch,#0FFH          ; one shot
;       jr      ProtectorSwitchDone            ;
;CountActive:
;       tcm     ProtectorSwitch,#03FH          ; test for the top
;       jr      z,ProtectorSwitchDone          ; if so skip
;       inc     ProtectorSwitch                ; set the next value
;       cp      ProtectorSwitch,#54d           ; test for too long
;       jr      nz,ProtectorSwitchDone         ; if not then done
;       ld      ProtectorSwitch,#0FFH          ; turn off till next pulse
;       jr      ProtectorSwitchDone            ;
;
;ZeroProtectorCounter:
;       clr     ProtectorSwitch                ; clear the counter
;ProtectorSwitchDone:
        srp     #LEARNEE_GRP                   ; set the register pointer
        dec     AOBSTEST                       ; decrease the aobs test timer
        jr      nz,NOFAIL                      ; if the timer not at 0 then it didnot fail
AOBSFAIL:
        ld      AOBSSTATUS,#0FFh               ; set the flag for a aobs
        ld      AOBSTEST,#11d                  ; if it failed reset the timer
        or      AOBSF,#00000001b               ; set the failed flag bit
NOFAIL:
        inc     t125ms                         ; increment the 125 mS timer
        tcm     T0EXT,#00000111B               ; test for the 111
        jp      nz,TEST125                     ; if not true then jump
FOURMS:
        cp      RPMONES,#00H                   ; test for the end of the one sec timer
        jr      z,TESTPERIOD                   ; if one sec over then test the pulses
                                               ; over the period
        dec     RPMONES                        ; else decrease the timer
        clr     RPM_COUNT                      ; start with a count of 0
        jr      RPMTDONE
```

```
TESTPERIOD:
        cp      RPMCLEAR,#00H           ; test the clear test timer for 0
        jr      nz,RPMTDONE             ; if not timed out then skip
        ld      RPMCLEAR,#122d          ; set the clear test time for next cycle .5
        cp      RPM_COUNT,#50d          ; test the count for too many pulses
        jr      ugt,FAREV               ; if too man pulses then reverse
        clr     RPM_COUNT               ; clear the counter
        jr      RPMTDONE                ; continue
FAREV:
        ld      FAULTCODE,#07h          ; set the fault flag
        ld      FAREVFLAG,#088H         ; set the forced up flag
        and     p0,#^LB ^C WORKLIGHT    ; turn off light
        ld      REASON,#80H             ; rpm forcing up motion
        call    SET_AREV_STATE          ; set the autorev state
RPMTDONE:
        dec     RPMCLEAR                ; decrement the timer
        cp      LIGHT1S,#00             ; test for the end
        jr      z,SKIPLIGHTE
        dec     LIGHT1S                 ; down count the light time
SKIPLIGHTE:
        inc     R_DEAD_TIME
        cp      RTO,#101D               ; test for the radio time out
        jr      ult,DONOTCB             ; if not timed out donot clear b
        clr     BCODEFLAG               ; else clear the b code flag
DONOTCB:

        cp      RsRto,#0FFH             ; inc to the ff position
        jr      z,SkipRsRtoInc          ;
        inc     RsRto
SkipRsRtoInc:


        inc     RTO                     ; increment the radio time out
        jr      nz,RTOOK                ; if the radio timeout ok then skip
        dec     RTO                     ; back turn
RTOOK:

TEST125:
        cp      t125ms,#125D            ; test for the time out
        jr      z,ONE25MS               ; if true the jump
        cp      t125ms,#63D             ; test for the other timeout
        jr      nz,N125
        call    FAULTB
        cp      RsTimer,#0FFH           ; test for the end of the rs232 period
        jr      z,SkipRs1TimerInc       ; if off skip increasing the counter
        inc     RsTimer                 ; increase the RsTimer till FF
        cp      RsTimer,#0FFH           ; test for the end of the rs232 period
        jr      z,SkipRs1TimerInc       ; if off skip increasing the counter
        inc     RsTimer                 ; increase the RsTimer till FF
        cp      RsTimer,#0FFH           ; test for the end of the rs232 period
        jr      z,SkipRs1TimerInc       ; if off skip increasing the counter
        inc     RsTimer                 ; increase the RsTimer till FF
        cp      RsTimer,#0FFH           ; test for the end of the rs232 period
        jr      z,SkipRs1TimerInc       ; if off skip increasing the counter
        inc     RsTimer                 ; increase the RsTimer till FF
SkipRs1TimerInc:
```

```
N125:
        pop     RP
        iret
ONE25MS:
        cp      RsTimer,#0FFH               ; test for the end of the rs232 period
        jr      z,SkipRs2TimerInc           ; if off skip increasing the counter
        inc     RsTimer                     ; increase the RsTimer till FF
        cp      RsTimer,#0FFH               ; test for the end of the rs232 period
        jr      z,SkipRs2TimerInc           ; if off skip increasing the counter
        inc     RsTimer                     ; increase the RsTimer till FF
        cp      RsTimer,#0FFH               ; test for the end of the rs232 period
        jr      z,SkipRs2TimerInc           ; if off skip increasing the counter
        inc     RsTimer                     ; increase the RsTimer till FF
        cp      RsTimer,#0FFH               ; test for the end of the rs232 period
        jr      z,SkipRs2TimerInc           ; if off skip increasing the counter
        inc     RsTimer                     ; increase the RsTimer till FF
SkipRs2TimerInc:
        inc     P8Counter                   ; increase the min time counter
        cp      P8Counter,#0d               ; ever 32 sec
        jr      nz,SkipTempStorage          ;
        inc     MinTimer                    ; increase timer
        tm      MinTimer,#00011111B         ; every 15 min
        jr      nz,SkipTempStorage          ;
        cp      MotorTempHi,PastTemp        ; test for the change
        jr      z,SkipTempStorage           ; if same do not change
        ld      PastTemp,MotorTempHi        ; save new value as past
        jr      nz,SkipTempStorage          ; store the temp in nonvol
        ld      STACKFLAG,#0AAH             ; save the temperature flag

SkipTempStorage:
        tm      P8Counter,#00000111B        ; every sec
        jr      nz,SkipTempOperation        ; if not at a sec skip
        cp      STATE,#1d                   ; test for the up direction
        jr      z,Running                   ; if so then running
        cp      STATE,#4d                   ; test for the down direction
        jr      z,Running                   ; if so then running
        tm      P8Counter,#01111111B        ; every 16 sec
        jr      nz,SkipTempOperation        ; if no then skip decreasing T
Idle:
        cp      MotorTempHi,Temperature     ; test for the min temp
        jr      ule,SkipTempOperation       ; if motor cool skip decrease
        ld      TDifference,MotorTempHi     ; read the motor temp and
        sub     TDifference,Temperature     ; subtract the
        sub     MotorTempLo,TDifference     ; decrease the temperature
        sbc     MotorTempHi,#00d
        sub     MotorTempLo,TDifference     ; decrease the temperature
        sbc     MotorTempHi,#00d
        jr      SkipTempOperation           ; done

Running:
        cp      FORCE_IGNORE,#00            ; test for past force ignore
        jr      nz,TestForStall             ; if not past test for a stall
AddRunningNumber:
        add     MotorTempLo,#TempRunIncLo   ; ADD the temp increase
        adc     MotorTempHi,#TempRunIncHi
        jr      SkipTempOperation
```

```
TestForStall:
        cp      RPM_ACOUNT,#02d                 ; test for any revs
        jr      uge,AddRunningNumber
AddStallNumber:
        add     MotorTempLo,#TempStallIncLo     ; ADD the temp increase
        adc     MotorTempHi,#TempStallIncHi

SkipTempOperation:
        cp      UpDown,#0FFH                    ; test for the max time
        jr      z,UpDownSkipInc                 ; if so dont inc
        inc     UpDown
UpDownSkipInc:
        inc     P5UTD                           ; increase the up to down flag
        call    FAULTB                          ; call the fault blinker
        clr     t125ms                          ; reset the timer
        inc     DOG2                            ; incrwease the second watch dog
        di
        inc     SDISABLE                        ; count off the system disable timer
        jr      nz,DO12                         ; if not rolled over then do the 1.2 sec
        dec     SDISABLE                        ; else reset to FF
DO12:
        cp      ONEP2,#00                       ; test for 0
        jr      z,INCLEARN                      ; if counted down then increment learn
        dec     ONEP2                           ; else down count
INCLEARN:
        inc     learnt                          ; increase the learn timer
        cp      learnt,#0H                      ; test for overflow
        jr      nz,LEARNTOK                     ; if not 0 skip back turning
        dec     learnt                          ;
LEARNTOK:
        ei
        inc     eraset                          ; increase the erase timer
        cp      eraset,#0H                      ; test for overflow
        jr      nz,ERASETOK                     ; if not 0 skip back turning
        dec     eraset                          ;
ERASETOK:
        pop     RP
        iret

;       fault blinker

FAULTB:
        inc     FAULTTIME                       ; increase the fault timer
        inc     FAULTTIME                       ; increase the fault timer
        cp      FAULTTIME,#090h                 ; test for the end
        jr      ult,FIRSTFAULT                  ; if not timed out
        clr     FAULTTIME                       ; reset the clock
        clr     FAULT                           ; clear the last
        cp      FAULTCODE,#4d                   ; test for over temp
        jr      nz,NotTempFault                 ; if not skip testing for clear
        cp      MotorTempHi,#DnSetMaxTemp       ; test for max temp
        jr      uge,NotTempFault                ; still hot donot clear
        clr     FAULTCODE
NotTempFault:
        cp      FAULTCODE,#04h                  ; test for call dealer code
        jr      UGE,GOTFAULT                    ; set the fault
```

```
TESTAOBSM:
        cp      STATE,#1d               ; test for door travel
        jr      z,NOAOBSFAULT           ; and if so skip fault code
        cp      STATE,#4d               ; test for door travel
        jr      z,NOAOBSFAULT           ; and if so skip fault code

        tm      AOBSF,#00000001b        ; test for the skiped aobs pulse
        jr      z,NOAOBSFAULT           ; if no skips then no faults
        tm      AOBSF,#00000010b        ; test for any pulses
        jr      z,NOPULSE               ; if no pulses find if hi or low
                                        ; else we are intermittent
        ld      FAULTCODE,#03h          ; set the fault
        jr      GOTFAULT                ; if same got fault
NOPULSE:
        tm      P3,#00000010b           ; test the input pin
        jr      nz,AOBSSH               ; jump if aobs is stuck hi
        cp      FAULTCODE,#01h          ; test for stuck low in the past
        jr      z,GOTFAULT              ; set the fault
        ld      FAULTCODE,#01h          ; set the fault code
        jr      FIRSTFC                 ;
AOBSSH:
        cp      FAULTCODE,#02h          ; test for stuck high in past
        jr      z,GOTFAULT              ; set the fault
        ld      FAULTCODE,#02h          ; set the code
        jr      FIRSTFC                 ;

GOTFAULT:
        ld      FAULT,FAULTCODE         ; set the code
        swap    FAULT                   ;
        jr      FIRSTFC                 ;
NOAOBSFAULT:
        clr     FAULTCODE               ; clear the fault code
FIRSTFC:
        clr     AOBSF                   ; clear flags

FIRSTFAULT:
        cp      FAULT,#00               ; test for no fault
        jr      z,NOFAULT
        ld      FAULTFLAG,#0FFH         ; set the fault flag
        cp      LEARNT,#0FFH            ; test for not in learn mode
        jr      nz,TESTSDI              ; if in learn then skip setting
        cp      FAULT,FAULTTIME         ;
        jr      ULE,TESTSDI

        tm      FAULTTIME,#00001000b    ; test the 1 sec bit
        jr      nz,BITONE
        ld      LearnLed,#01000000B     ; turn on the led
        ret

BITONE:
        ld      LearnLed,#01111111B     ; turn off the led
TESTSDI:
        ret

NOFAULT:
        clr     FAULTFLAG               ; clear the flag
```

```
        tm      LearnLed,#01000000B          ; test for fault blink on
        jr      z,LeaveLedSet
        ld      LearnLed,#00111111b          ; turn off the led

LeaveLedSet:
        ret

;-------------------------------------------------------------------
; MOTOR STATE MACHINE
;-------------------------------------------------------------------

STATEMACHINE:
        xor     p0,#00001000b                ; toggle aux output
        cp      DOG2,#8d                     ; test the 2nd watchdog for problem
        jp      ugt,START                    ; if problem reset
        cp      STATE,#06d                   ; test for legal number
        jp      ugt,start                    ; if not the reset
        jp      z,stop                       ; stop motor          6
        cp      STATE,#03d                   ; test for legal number
        jp      z,start                      ; if not the reset
        cp      STATE,#00d                   ; test for autorev
        jp      z,auto_rev                   ; auto reversing      0
        cp      STATE,#01d                   ; test for up
        jp      z,up_direction               ; door is going up    1
        cp      STATE,#02d                   ; test for autorev
        jp      z,up_position                ; door is up          2
        cp      STATE,#04d                   ; test for autorev
        jp      z,dn_direction               ; door is going down  4
        jp      dn_position                  ; door is down        5

;-------------------------------------------------------------------
; AUX OBSTRUCTION OUTPUT AND LIGHT FUNCTION
;-------------------------------------------------------------------

AUXLIGHT:
test_light_on:
        cp      LIGHT_FLAG,#LIGHT            ;
        jr      z,dec_pre_light              ;
        cp      LIGHT1S,#00                  ; test for no flash
        jr      z,NO1S                       ; if not skip
        cp      LIGHT1S,#01d                 ; test for timeout
        jr      nz,NO1S                      ; if not skip
        xor     p0,#WORKLIGHT                ; toggle light
        clr     LIGHT1S                      ; oneshoted
NO1S:
        cp      FLASH_FLAG,#FLASH
        jr      nz,dec_pre_light             ;
        decw    FLASH_DELAY                  ; 250 ms period
        jr      nz,dec_pre_light             ;
        xor     p0,#WORKLIGHT                ; toggle light
        ld      FLASH_DELAY_HI,#FLASH_HI
        ld      FLASH_DELAY_LO,#FLASH_LO
        dec     FLASH_COUNTER                ;
        jr      nz,dec_pre_light
        clr     FLASH_FLAG                   ;
```

```
dec_pre_light:
        cp      LIGHT_TIMER_HI,#0FFH            ; test for the timer ignore
        jr      z,exit_light                    ; if set then ignore
        dec     PRE_LIGHT                       ;  dec 3 byte light timer
        jr      nz,exit_light                   ;
        decw    LIGHT_TIMER                     ;
        jr      nz,exit_light                   ; if timer 0 turn off the light
        and     p0,#^C LIGHT_ON                 ; turn off the light
exit_light:
        ret                                     ; return

;-----------------------------------------------------------------------
;                       AUTO_REV ROUTINE
;-----------------------------------------------------------------------

auto_rev:
        cp      FAREVFLAG,#088H                 ; test for the forced up flag
        jr      nz,LEAVEREV
        and     p0,#^LB ^C WORKLIGHT             ; turn off light
LEAVEREV:
        .IF     E21
        xor     P1,#00000001B                   ; Kick the external dog
        .ELSE
        WDT                                     ; KICK THE DOG
        .ENDIF
        call    HOLDFREV                        ; hold off the force reverse
        ld      LIGHT_FLAG,#LIGHT               ; force the light on no blink
        and     p0,#^LB ^C MOTOR_UP ^& #^C MOTOR_DN    ; disable motor
        di
        decw    AUTO_DELAY                      ; wait for .5 second
        decw    BAUTO_DELAY                     ; wait for .5 second
        ei
        jr      nz,arswitch                     ; test switches


        or      p0,#00001000b                   ; set aux output for FEMA
        ld      REASON,#40H                     ; set the reason for the change
        jp      SetUpDirStateNoTemp             ; set the state
arswitch:
        cp      WIN_FLAG,#00h                   ; test for window active
        jr      z,exit_auto_rev                 ; if inactive skip commands
        ld      REASON,#00H                     ; set the reason to command
        cp      SW_DATA,#CMD_SW                 ; test for a command
        jp      z,SET_STOP_STATE                ; if so then stop
        ld      REASON,#10H                     ; set the reason as radio command
        cp      RADIO_CMD,#0AAH                 ; test for a radio command
        jp      z,SET_STOP_STATE                ; if so the stop
exit_auto_rev:
        ret                                     ; return


HOLDFREV:
        ld      RPMONES,#244d                   ; set the hold off
        ld      RPMCLEAR,#122d                  ; clear rpm reverse .5 sec
        clr     RPM_COUNT                       ; start with a count of 0
        ret
```

```
;-----------------------------------------------------------------------
;                  DOOR GOING UP
;-----------------------------------------------------------------------

up_direction:
        .IF     E21
        xor     P1,#00000001B               ; Kick the external dog
        .ELSE
        WDT                                 ; KICK THE DOG
        .ENDIF
        cp      OnePass,STATE               ; test for memory read yet
        jr      z,UpContinue
        ret
UpContinue:
        call    HOLDFREV                    ; hold off the force reverse
        ld      LIGHT_FLAG,#LIGHT           ; force the light on no blink
        and     p0,#^LB ^C MOTOR_DN         ; disable down relay

        cp      MOTDEL,#0FFH                ; test for done
        jr      z,UPON                      ; if done skip delay
        inc     MOTDEL                      ; increase the delay timer
        or      p0,#LIGHT_ON                ; turn on the light
        cp      MOTDEL,#20d                 ; test for 40 seconds
        jr      ule,UPOFF                   ; if not timed
UPON:
        or      p0,#MOTOR_UP ^| #LIGHT_ON   ; turn on the motor and light
UPOFF:
        cp      FORCE_IGNORE,#01            ; test fro the end of the force ignore
        jr      nz,SKIPUPRPM                ; if not donot test rpmcount
        cp      RPM_ACOUNT,#02H             ; test for less the 2 pulses
        jr      ugt,SKIPUPRPM               ;
        ld      FAULTCODE,#06h
SKIPUPRPM:
        cp      FORCE_IGNORE,#00            ; test timer for done
        jr      nz,test_up_sw_pre           ; if timer not up do not test force
TEST_UP_FORCE:
        di
        dec     RPM_TIME_OUT                ; decrease the timeout
        dec     BRPM_TIME_OUT               ; decrease the timeout
        ei
        jr      z,failed_up_rpm
        di                                  ; turn off the interrupt
        push    UP_FORCE_LO                 ; save the force setting
        push    UP_FORCE_HI
        sub     UP_FORCE_LO,RPM_PERIOD_LO
        sbc     UP_FORCE_HI,RPM_PERIOD_HI
        tm      UP_FORCE_HI,#10000000B      ; test high bit for sign
        jr      z,test_up_sw_pop            ; if the rpm period is ok then switch
        pop     UP_FORCE_HI                 ; reset the force setting
        pop     UP_FORCE_LO                 ;
        ei
failed_up_rpm:
        ld      REASON,#20H                 ; set the reason as force
        jp      SET_STOP_STATE
```

```
test_up_sw_pre:
        dec     FORCE_PRE                       ; dec the prescaler
        tm      FORCE_PRE,#00000001B            ; test for odd /2
        jr      nz,test_up_sw                   ; if odd skip
        di
        dec     FORCE_IGNORE
        dec     BFORCE_IGNORE
        jr      test_up_sw                      ;
test_up_sw_pop:
        pop     UP_FORCE_HI                     ; reset the force setting
        pop     UP_FORCE_LO                     ;
        ei
test_up_sw:
        ei                                      ; enable interrupt
        cp      L_A_C,#044H                     ; test for learning up limit
        jr      z,get_sw                        ; if so skip testing the limit
        cp      POSITION_HI,#07FH               ; test for the middle range
        jr      nz,TESTUPN                      ; if not test the up limit normal
        cp      POSITION_LO,#00                 ; test for the limit
        jr      z,UPLIM                         ; if so then jump
TESTUPN:

        di
        push    POSITION_LO
        push    POSITION_HI
        sub     POSITION_LO,UP_LIM_LO           ; find the difference from position
        sbc     POSITION_HI,UP_LIM_HI           ;
        cp      POSITION_HI,#0FFH               ; test for a within 256 of after limit
        jr      z,UP_LIM_SET

        pop     POSITION_HI                     ; reset the position
        pop     POSITION_LO
        ei
        jr      get_sw                          ; if not at the limit test switches
UP_LIM_SET:
        pop     POSITION_HI                     ; reset the position
        pop     POSITION_LO
        ei


UPLIM:
        ld      REASON,#50H                     ; set the reason as limit
        jp      SET_UP_POS_STATE                ;
get_sw:
        cp      WIN_FLAG,#00h                   ; test for the flag active
        jr      z,test_up_time                  ; if inactive skip command
        ld      REASON,#10H                     ; set the radio command reason
        cp      RADIO_CMD,#0AAH                 ;  test for a radio command
        jp      z,SET_STOP_STATE                ; if so stop
        ld      REASON,#00H                     ; set the reason as a command
        cp      SW_DATA,#CMD_SW                 ;  test for a command condition
        jr      ne,test_up_time                 ;
        jp      SET_STOP_STATE                  ;
test_up_time:
        ld      REASON,#70H                     ; set the reason as a time out
        decw    MOTOR_TIMER                     ; decrement motor timer
```

```
        jp      z,SET_STOP_STATE                        ;
exit_up_dir:
        ret                                             ; return to caller
;-----------------------------------------------------------------------
;               DOOR UP
;-----------------------------------------------------------------------

up_position:
        .IF     E21
        xor     P1,#00000001B                           ; Kick the external dog
        .ELSE
        WDT                                             ; KICK THE DOG
        .ENDIF
        cp      FAREVFLAG,#088H                         ; test for the forced up flag
        jr      nz,LEAVELIGHT
        and     p0,#^LB ^C WORKLIGHT                    ; turn off light
        jr      UPNOFLASH                               ; skip clearing the flash flag
LEAVELIGHT:
        ld      LIGHT_FLAG,#00H                         ; allow blink
UPNOFLASH:
        and     p0,#^LB ^C MOTOR_UP ^& #^C MOTOR_DN     ; disable motor
        cp      SW_DATA,#LIGHT_SW                       ; light sw debounced?
        jr      z,work_up                               ;
        cp      UpDown,#UpDownTime                      ; test for the direction delay
        jr      ult,UpPosRet
        ld      REASON,#10H                             ; set the reason as  a radio command
        cp      RADIO_CMD,#0AAH                         ; test for a radio cmd
        jr      z,SETDNDIRSTATE                         ; if so start down
        ld      REASON,#00H                             ; set the reason as a command
        cp      SW_DATA,#CMD_SW                         ; command sw debounced?
        jr      z,SETDNDIRSTATE                         ; if command
UpPosRet:
        ret
SETDNDIRSTATE:
        ld      ONEP2,#10D                              ; set the 1.2 sec timer
        jp      SET_DN_DIR_STATE

work_up:
        clr     SW_DATA
        xor     p0,#WORKLIGHT                           ; toggle work light
        ld      LIGHT_TIMER_HI,#0FFH                    ; set the timer ignore
up_pos_ret:
        ret                                             ; return
;-----------------------------------------------------------------------
;               DOOR GOING DOWN
;-----------------------------------------------------------------------

dn_direction:


        .IF     E21
        xor     P1,#00000001B                           ; Kick the external dog
        .ELSE
        WDT                                             ; KICK THE DOG
        .ENDIF
        cp      OnePass,STATE                           ; test for memory read yet
```

```
        jr      z,DownContinue
        ret
DownContinue:
        cp      L_A_C,#044H                 ; Durring setup move the
        jr      ule,NORM_DN                 ; present position into the
        push    rp                          ; limit while traveling down
        srp     #FORCE_GRP
        .IF     P5BlockFlag
        ld      DN_LIM_HI,position_hi       ;
        ld      DN_LIM_LO,position_lo       ;
        tm      P0,#00100000B               ; test for 10-9.5  or 8-6
        jr      nz,L86                      ; gear reduction
L109P5
        tm      P0,#00010000B               ; test for 10 vs 9.5
        jr      nz,L9P5                     ;
L10:
        sub     DN_LIM_LO,#L10Lo            ; subtract .5 inches
        sbc     DN_LIM_HI,#L10Hi
        jr      GotLimitPosition
L9P5:
        sub     DN_LIM_LO,#L9P5Lo           ; subtract .5 inches
        sbc     DN_LIM_HI,#L9P5Hi
        jr      GotLimitPosition
L86:
        tm      P0,#00010000B               ; test for 10 vs 9.5
        jr      nz,L8                       ;
L6:
        sub     DN_LIM_LO,#L6Lo             ; subtract .5 inches
        sbc     DN_LIM_HI,#L6Hi
        jr      GotLimitPosition
L8:
        sub     DN_LIM_LO,#L8Lo             ; subtract .5 inches
        sbc     DN_LIM_HI,#L8Hi
        jr      GotLimitPosition

        .ELSE
        ld      DN_LIM_HI,position_hi       ;
        ld      DN_LIM_LO,position_lo       ;
        .ENDIF
GotLimitPosition:
        pop     rp
NORM_DN:
        call    HOLDFREV                    ; hold off the force reverse
        clr     FLASH_FLAG                  ; turn off the flash
        ld      LIGHT_FLAG,#LIGHT           ; force the light on no blink
        and     p0,#^LB ^C MOTOR_UP         ; turn off motor up
        cp      MOTDEL,#0FFH                ; test for done
        jr      z,DNON                      ; if done skip delay
        inc     MOTDEL                      ; increase the delay timer
        or      p0,#LIGHT_ON                ; turn on the light
        cp      MOTDEL,#20d                 ; test for 40 seconds
        jr      ule,DNOFF                   ; if not timed
DNON:

        or      p0,#MOTOR_DN ^| #LIGHT_ON   ; turn on the motor and light
DNOFF:
```

```
        cp      FORCE_IGNORE,#01          ; test fro the end of the force ignore
        jr      nz,SKIPDNRPM              ; if not donot test rpmcount
        cp      RPM_ACOUNT,#02H           ; test for less the 2 pulses
        jr      ugt,SKIPDNRPM             ;
        ld      FAULTCODE,#06h
SKIPDNRPM:
        cp      FORCE_IGNORE,#00          ; test timer for done
        jr      nz,test_dn_sw_pre         ; if timer not up do not test force
TEST_DOWN_FORCE:
        di
        dec     RPM_TIME_OUT              ; decrease the timeout
        dec     BRPM_TIME_OUT             ; decrease the timeout
        ei
        jr      z,failed_dn_rpm
        di
        push    DN_FORCE_LO               ; save the value
        push    DN_FORCE_HI
        sub     DN_FORCE_LO,RPM_PERIOD_LO
        sbc     DN_FORCE_HI,RPM_PERIOD_HI
        tm      DN_FORCE_HI,#10000000B    ; test high bit for sign
        jr      z,test_dn_sw_pop          ; if the rpm period is ok then switch
        pop     DN_FORCE_HI               ; reset the value
        pop     DN_FORCE_LO
        ei
failed_dn_rpm:
        cp      L_A_C,#47h                ; test for the state for storage
        jr      nz,NoStoreDown            ; if not then continue
        cp      AOBS_FLAG,#01h            ; test for the pass point set
        jr      z,NoStoreDown             ; if passed donot set the limit
        cp      STATE,#00                 ; test for past state 0
        jr      nz,NoStoreDown            ; if past 0 donot set the limit
StoreUpLimError:
        clr     UP_LIM_HI                 ;
        clr     UP_LIM_LO
        sub     UP_LIM_LO,position_lo     ; get the - of the count
        sbc     UP_LIM_HI,position_hi     ;
        call    FIND_WINDOW               ; find the window
NoStoreDown:
        ld      REASON,#20H               ; set the reason as force
        jp      SET_AREV_STATE            ; set the state
test_dn_sw_pre:
        dec     FORCE_PRE                 ; dec the prescaler
        tm      FORCE_PRE,#00000001B      ; test for odd /2
        jr      nz,test_dn_sw             ; if odd skip
        di
        dec     FORCE_IGNORE
        dec     BFORCE_IGNORE
        jr      test_dn_sw
test_dn_sw_pop:
        pop     DN_FORCE_HI               ; reset the value
        pop     DN_FORCE_LO
        ei
test_dn_sw:
        ei                                ; turn on the interrupt
        cp      L_A_C,#044H               ; test for the auto position setting
        jr      ugt,call_sw_dn            ; if so skip testing limit
```

```
        cp      AOBSSTATE,#00               ; test for looking at the zeroer
        jr      nz,call_sw_dn               ;

        di
        push    POSITION_LO                 ; save the position
        push    POSITION_HI
        sub     POSITION_LO,DN_LIM_LO       ; find the difference from position
        sbc     POSITION_HI,DN_LIM_HI       ;
        cp      POSITION_HI,#00             ; test for a within 256 of after limit
        jr      z,DN_LIM_SET

        pop     POSITION_HI                 ; reset the position
        pop     POSITION_LO
        ei
        jr      call_sw_dn                  ; if not at the limit test radio
DN_LIM_SET:
        pop     POSITION_HI                 ; reset the position
        pop     POSITION_LO
        ei
DOWNLIM:

        .IF     DownToLimits

        cp      CMD_DEB,#0FFH               ; test for the command held
        jr      z,dn_lim_stop               ; if so skip aobs

        .ENDIF

        cp      AOBSSTATE,#00               ; test for the finish of the counter
        jr      nz,AOBSFUNCTION             ; AOBS happened near the limit
        cp      AOBS_FLAG,#00               ; test for the flag for pass point
        jr      z,AOBSERROR                 ; error reverse
dn_lim_stop:
        ld      REASON,#50H                 ; set the reason as a limit
        cp      CMD_DEB,#0FFH               ; test for the switch still held
        jr      nz,TESTRADIO                ;
        ld      REASON,#90H                 ; closed with the control held
        jr      TESTFORCEIG
TESTRADIO:
        cp      LAST_CMD,#00                ; test for the last command being radio
        jr      nz,TESTFORCEIG              ; if not test force
        cp      BCODEFLAG,#077H             ; test for the b code flag
        jr      nz,TESTFORCEIG              ;
        ld      REASON,#0A0H                ; set the reason as b code to limit
TESTFORCEIG:
        cp      FORCE_IGNORE,#00H           ; test the force ignore for done
        jr      z,NOAREVDN                  ; a rev if limit before force enabled
        ld      REASON,#60h                 ; early limit
        jp      SET_AREV_STATE              ; set autoreverse
NOAREVDN:
        and     p0,#^LB ^C MOTOR_DN         ;
        jp      SET_DN_POS_STATE            ; set the state
call_sw_dn:
        cp      WIN_FLAG,#00h               ; test for window active
        jr      z,test_dn_time              ; if inactive then skip command
        ld      REASON,#10H                 ; set the reason as radio command
```

```
        cp      RADIO_CMD,#0AAH                 ; test for a radio command
        jp      z,SET_AREV_STATE                ; if so arev
        ld      REASON,#00H                     ; set the reason as command
        cp      SW_DATA,#CMD_SW                 ;  test for command
        jp      z,SET_AREV_STATE                ;
test_dn_time:
        ld      REASON,#70H                     ; set the reason as timeout
        decw    MOTOR_TIMER                     ; decrement motor timer
        jp      z,SET_AREV_STATE                ;
        cp      OBS_FLAG,#0CCH                  ; test the flag for count
        jr      nz,exit_dn_dir                  ; if not then exit
AOBSFUNCTION:
        .IF     AOBSBypass                      ; if the aobs can be bypassed from
                                                ; a held command or held B code
        cp      LAST_CMD,#00                    ; test for the last command from radio
        jr      z,OBSTESTB                      ; if last command was a radio test b
        cp      CMD_DEB,#0FFH                   ; test for the command switch holding
        jr      nz,OBSAREV                      ; if the command switch is not holding
                                                ; do the autorev
        ret                                     ; otherwise skip
        .ENDIF
OBSAREV:
        ld      FLASH_FLAG,#0FFH                ; set flag
        ld      FLASH_COUNTER,#20               ; set for 10 flashes
        ld      FLASH_DELAY_HI,#FLASH_HI        ; set for .5 Hz period
        ld      FLASH_DELAY_LO,#FLASH_LO
        ld      REASON,#30H                     ; set the reason as autoreverse
        jp      SET_AREV_STATE                  ;
OBSTESTB:
        cp      BCODEFLAG,#077H                 ; test for the b code flag
        jr      nz,OBSAREV                      ; if not b code then arev
exit_dn_dir:
        ret                                     ; return
AOBSERROR:
        ld      REASON,#0F0h                    ; set the reason as no pass point
        jp      SET_AREV_STATE                  ;

;-----------------------------------------------------------------------
;               DOOR DOWN
;-----------------------------------------------------------------------

dn_position:
        .IF     E21
        xor     P1,#00000001B                   ; Kick the external dog
        .ELSE
        WDT                                     ; KICK THE DOG
        .ENDIF
        cp      FAREVFLAG,#088H                 ; test for the forced up flag
        jr      nz,DNLEAVEL                     ;
        and     p0,#^LB ^C WORKLIGHT            ; turn off light
        jr      DNNOFLASH                       ; skip clearing the flash flag
DNLEAVEL:
        ld      LIGHT_FLAG,#00H                 ; allow blink
DNNOFLASH:
        and     p0,#^LB ^C MOTOR_UP ^& #^C MOTOR_DN    ; disable motor
        cp      SW_DATA,#LIGHT_SW               ; debounced? light
```

```
        jr      z,work_dn                         ;
        cp      UpDown,#UpDownTime                ; test for the .5 seconds direction
        jr      ult,DnPosRet

        ld      REASON,#10H                       ; set the reason as a radio command
        cp      RADIO_CMD,#0AAH                   ; test for a radio command
        jr      z,SETUPDIRSTATE                   ; if so go up
        ld      REASON,#00H                       ; set the reason as a command
        cp      SW_DATA,#CMD_SW                   ; command sw pressed?
        jr      z,SETUPDIRSTATE                   ; if so go up
DnPosRet:
        ret

SETUPDIRSTATE:
        ld      ONEP2,#10D                        ; set the 1.2 sec timer
        jp      SET_UP_DIR_STATE

work_dn:
        clr     SW_DATA
        clr     RADIO_CMD
        xor     p0,#WORKLIGHT                     ; toggle work light
        ld      LIGHT_TIMER_HI,#0FFH              ; set the timer ignore
dn_pos_ret:
        ret                                       ; return
;-----------------------------------------------------------------------
;               STOP
;-----------------------------------------------------------------------

stop:
        .IF     E21
        xor     P1,#00000001B                     ; Kick the external dog
        .ELSE
        WDT                                       ; KICK THE DOG
        .ENDIF
        cp      FAREVFLAG,#088H                   ; test for the forced up flag
        jr      nz,LEAVESTOP
        and     p0,#^LB ^C WORKLIGHT              ; turn off light
LEAVESTOP:
        ld      LIGHT_FLAG,#00H                   ; allow blink
        and     p0,#^LB ^C MOTOR_UP ^& #^C MOTOR_DN    ; disable motor
        cp      SW_DATA,#LIGHT_SW                 ; debounced? light
        jr      z,work_stop                       ;
        cp      UpDown,#UpDownTime                ; test for the .5 seconds direction
        jr      ult,StopPosRet

        ld      REASON,#10H                       ; set the reason as radio command
        cp      RADIO_CMD,#0AAH                   ; test for a radio command
        jp      z,SET_DN_DIR_STATE                ; if so go down
        ld      REASON,#00H                       ; set the reason as a command
        cp      SW_DATA,#CMD_SW                   ; command sw pressed?
        jp      z,SET_DN_DIR_STATE                ; if so go down
StopPosRet:
        ret
work_stop:
        clr     SW_DATA
        clr     RADIO_CMD
```

```
        xor     p0,#WORKLIGHT                   ; toggle work light
        ld      LIGHT_TIMER_HI,#0FFH            ; set the timer ignore
stop_ret:
        ret                                     ; return

;-----------------------------------------------------------------------
;       SET THE AUTOREV STATE
;-----------------------------------------------------------------------
SET_AREV_STATE:
        clr     SW_DATA                         ; clear the switch data
        clr     RADIO_CMD                       ; clear the radio command
        di
        cp      L_A_C,#47H                      ; test for the store force data
        jr      nz,NOSD
        add     P32_MAX_LO,ForceAddLo           ; ADD the force adder
        adc     P32_MAX_HI,ForceAddHi
        ld      DN_FORCE_HI,P32_MAX_HI          ; transfer the force
        ld      DN_FORCE_LO,P32_MAX_LO
NOSD:
        ld      STATE,#AUTO_REV                 ; if we got here, then reverse motor
        ld      BSTATE,#AUTO_REV                ; if we got here, then reverse motor
        ei
        jp      SET_ANY

;-----------------------------------------------------------------------
;       SET THE STOPPED STATE
;-----------------------------------------------------------------------

Temp_SET_STOP_STATE:
        ld      FAULTCODE,#04d                  ; set the fault blink
        jr      SetStopStateNoWrite

Mem_SET_STOP_STATE:
        ld      FAULTCODE,#05D                  ; set the fault blink

SetStopStateNoWrite:
        ld      MinTimer,#01D                   ; set next write min out
        clr     SW_DATA                         ; clear the switch data
        clr     RADIO_CMD                       ; clear the radio command

        di
        ld      STATE,#STOP                     ;
        ld      BSTATE,#STOP                    ;
        ei
        jp      SetAnyNoWrite


SET_STOP_STATE:

        ld      MinTimer,#01D                   ; set next write min out
        clr     SW_DATA                         ; clear the switch data
        clr     RADIO_CMD                       ; clear the radio command

        di
        ld      STATE,#STOP                     ;
```

```
        ld      BSTATE,#STOP                        ;
        ei
        jp      SET_ANY

;-----------------------------------------------------------------
;       SET THE DOWN DIRECTION STATE
;-----------------------------------------------------------------
SET_DN_DIR_STATE:
        clr     SW_DATA                             ; clear the switch data
        clr     RADIO_CMD                           ; clear the radio command
        call    TempMeasure                         ; measure the temperature
        di

        .IF     ThermalProtectorFlag

        tm      P2,#10000000B                       ; test for the switch state
        jr      z,SkipDownThermalProtector          ; skip if switch gnded
        ld      REASON,#0B0H                        ; set the reason as thermal
        cp      MotorTempHi,#DnSetMaxTemp           ; test if we need to skip for max temp
        jr      uge,Temp_SET_STOP_STATE

        .ENDIF
SkipDownThermalProtector:
        ld      STATE,#DN_DIRECTION                 ; energize door
        ld      BSTATE,#DN_DIRECTION                ; energize door
        ei
        clr     FAREVFLAG                           ; one shot the forced reverse

        cp      L_A_C,#042h                         ; test for learning the force and limits
        jp      UGE,SET_ANY                         ; if so then set the direction to down
        cp      DN_LIM_HI,#00h                      ; test for stuck bits
        jr      nz,TestSetDownBits                  ;
        cp      DN_LIM_LO,#00h                      ; test for stuck bits
        jr      nz,TestSetDownBits                  ;
        jp      Mem_SET_STOP_STATE                  ; if the bits are stuck then stop unit
TestSetDownBits:
        cp      DN_LIM_HI,#0FFh                     ; test for stuck bits
        jr      nz,DownBitsOk                       ;
        cp      DN_LIM_LO,#0FFh                     ; test for stuck bits
        jr      nz,DownBitsOk                       ;
        jp      Mem_SET_STOP_STATE                  ; if the bits are stuck then stop unit
DownBitsOk:
        cp      FAULTCODE,#5d                       ; test for memory fault
        jr      nz,DnSkipMemFaultClear              ; if so then clear
        clr     FAULTCODE
DnSkipMemFaultClear:
        di
        push    DN_LIM_HI                           ; save the limits
        push    DN_LIM_LO
        sub     DN_LIM_LO,POSITION_LO               ; find the difference from position
        sbc     DN_LIM_HI,POSITION_HI               ;
        cp      DN_LIM_HI,#00                       ; test for a 256 < number
        jr      z,POS_DN_LIM
        pop     DN_LIM_LO                           ; reset the limit
        pop     DN_LIM_HI
        ei
```

```
        jp      SET_ANY
POS_DN_LIM:
                                        ; reverse the direction if too close
                                        ; to the down limit
        pop     DN_LIM_LO               ; reset the limit
        pop     DN_LIM_HI
        ei
        jr      SetUpDirStateNoTemp

;-----------------------------------------------------------------------
;       SET THE UP DIRECTION STATE
;-----------------------------------------------------------------------
SET_UP_DIR_STATE:
        call    TempMeasure             ; measure the temperature
SetUpDirStateNoTemp:
        clr     SW_DATA                 ; clear the switch data
        clr     RADIO_CMD               ; clear the radio command
        di

        .IF     ThermalProtectorFlag
        tm      P2,#10000000B           ; test for the switch state
        jr      z,SkipUpThermalProtector ; skip if switch gnded

        cp      STATE,#AUTO_REV         ; if the state is autoreverse allow up
        jr      z,SkipUpThermalProtector
        ld      REASON,#0B0H            ; set the reason as thermal
        cp      MotorTempHi,#UpSetMaxTemp ; test if we need to skip for max temp
        jp      uge,Temp_SET_STOP_STATE

        .ENDIF
SkipUpThermalProtector:
        ld      STATE,#UP_DIRECTION     ;
        ld      BSTATE,#UP_DIRECTION    ;
        ei
        cp      L_A_C,#042H             ; test for learning the limits
        jr      UGE,SET_ANY             ; skip testing the limit if learning
RefreshUpLimit:
        cp      UP_LIM_HI,#00h          ; test for stuck bits
        jr      nz,TestSetUpBits
        cp      UP_LIM_LO,#00h          ; test for stuck bits
        jr      nz,TestSetUpBits
        jp      Mem_SET_STOP_STATE      ; if the bits are stuck then stop unit
TestSetUpBits:
        cp      UP_LIM_HI,#0FFh         ; test for stuck bits
        jr      nz,UpBitsOk
        cp      UP_LIM_LO,#0FFh         ; test for stuck bits
        jr      nz,UpBitsOk
        jp      Mem_SET_STOP_STATE      ; if the bits are stuck then stop unit
UpBitsOk:
        cp      FAULTCODE,#5d           ; test for memory fault
        jr      nz,UpSkipMemFaultClear  ; if so then clear
        clr     FAULTCODE
UpSkipMemFaultClear:
        jr      SET_ANY                 ; set the direction
```

```
;-----------------------------------------------------------------------
;       SET THE UP POSITION STATE
;-----------------------------------------------------------------------
SET_UP_POS_STATE:

        clr     SW_DATA                         ; clear the switch data
        clr     RADIO_CMD                       ; clear the radio command
        ld      MinTimer,#01D                   ; set next write min out

        di
        cp      L_A_C,#49h                      ; test for the store
        jr      nz,UPNS

        add     P32_MAX_LO,ForceAddLo           ; ADD the adder
        adc     P32_MAX_HI,ForceAddHi
        ld      UP_FORCE_HI,P32_MAX_HI          ; transfer the force
        ld      UP_FORCE_LO,P32_MAX_LO
UPNS:
        ld      STATE,#UP_POSITION              ;
        ld      BSTATE,#UP_POSITION             ;
        ei
        jr      SET_ANY

;-----------------------------------------------------------------------
;       SET THE DOWN POSITION STATE
;-----------------------------------------------------------------------
SET_DN_POS_STATE:

        clr     SW_DATA                         ; clear the switch data
        clr     RADIO_CMD                       ; clear the radio command
        ld      MinTimer,#01D                   ; set next write min out

        di
        ld      STATE,#DN_POSITION              ; load new state
        ld      BSTATE,#DN_POSITION             ; load new state
        ei
        cp      WIN_FLAG,#00                    ; test for the win
        jr      nz,SET_ANY                      ; if on skip
        inc     WIN_FLAG                        ; else turn on the window
        jr      SET_ANY


;-----------------------------------------------------------------------
;       SET ANY STATE
;-----------------------------------------------------------------------

SET_ANY:
        clr     UpDown                          ; clear the direction timer
        ld      STACKFLAG,#0FFH                 ; set the flag
SetAnyNoWrite:
        cp      L_A_C,#42H                      ; test for in learn mode
        jr      uge,SkipReadAny                 ; if so skip reading force

SkipReadAny:
        clr     AOBS_FLAG                       ; clear the flag
        clr     AOBSF                           ; clear any pending faults
```

```
        clr     AOBSSTATE                       ; reset the state counter
        clr     AOBSRPM                         ; clear any past aobs count
        clr     OBS_FLAG                        ;
        clr     AOBSB
        cp      L_A_C,#4CH                      ; test for learing down dir
        jr      z,SkipForceClear
        clr     MAX_F_HI                        ; clear the force reading
        clr     MAX_F_LO                        ;
        clr     P32_MAX_LO                      ;
        clr     P32_MAX_HI
SkipForceClear:
        clr     SW_DATA                         ; clear the switch data
        inc     L_A_C                           ; set the LAC to the next state
        di
        clr     RPM_COUNT                       ; clear the rpm counter
        ld      AUTO_DELAY_HI,#AUTO_HI          ; set the .5 second auto rev timer
        ld      AUTO_DELAY_LO,#AUTO_LO          ;
        ld      BAUTO_DELAY_HI,#AUTO_HI         ; set the .5 second auto rev timer
        ld      BAUTO_DELAY_LO,#AUTO_LO         ;
        ld      FORCE_IGNORE,#ONE_SEC           ; set the force ignore timer to one sec
        ld      BFORCE_IGNORE,#ONE_SEC          ; set the force ignore timer to one sec
        ei
ClearRadioCmd:
        clr     RADIO_CMD                       ; one shot
        clr     RPM_ACOUNT                      ; clear the rpm active counter
        ld      LIGHT_TIMER_HI,#SET_TIME_HI     ; set the light period
        ld      LIGHT_TIMER_LO,#SET_TIME_LO     ;
        ld      PRE_LIGHT,#SET_TIME_PRE         ;
        ld      MOTOR_TIMER_HI,#MOTOR_HI
        ld      MOTOR_TIMER_LO,#MOTOR_LO
        ld      STACKREASON,REASON              ; save the temp reason
        ld      LIGHTS,P0                       ; read the light state
        and     LIGHTS,#WORKLIGHT               ;
        jr      nz,lighton                      ; if the light is on skip clearing
lightoff:
        clr     MOTDEL                          ; clear the motor delay
lighton:
        ret


;-----------------------------------------------------------------
;               THIS THE AUXILARY OBSTRUCTION INTERRUPT ROUTINE
;-----------------------------------------------------------------

AUX_OBS:
        .IF E21
        and     imr,#11111011b                  ; turn off the interupt for up to 500uS
        .ELSE
        and     imr,#11110111b                  ; turn off the interupt for up to 500uS
        .ENDIF
        ld      AOBSTEST,#11D                   ; reset the test timer
        or      AOBSF,#00000010B                ; set the flag for got a aobs
        clr     AOBSSTATUS                      ; clear the aobs set state
        iret                                    ; return from int

;-----------------------------------------------------------------
```

```
;                  THIS IS THE MOTOR RPM INTERRUPT ROUTINE
;
;                  Direction for counter is the LSB of the state
;
;-----------------------------------------------------------------------

RPM:                                                        ; motor speed
         push    rp                                         ; save current pointer
         srp     #RPM_GROUP                                 ;point to these reg
         ld      rpm_temp_hi,T0EXT                          ; read the timer extension
         ld      rpm_temp_lo,T0                             ; read the timer
         tm      IRQ,#00010000B                             ; test for a pending interrupt
         jr      z,RPMTIMEOK                                ; if not then time ok
RPMTIMEERROR:
         tm      rpm_temp_lo,#10000000B                     ; test for timer reload
         jr      z,RPMTIMEOK                                ; if no reload time is ok
         dec     rpm_temp_hi                                ; if reloaded then dec the hi to resync
RPMTIMEOK:
         .IF E21
         and     imr,#11110111b                             ; turn off the interupt for up to 500uS
         .ELSE
         and     imr,#11111011b                             ; turn off the interupt for up to 500uS
         .ENDIF

         ld      rpm_2past_hi,rpm_past_hi                   ; save the past for testing
         ld      rpm_2past_lo,rpm_past_lo                   ;
         ld      rpm_past_hi,rpm_temp_hi                    ; transfer the present into the past
         ld      rpm_past_lo,rpm_temp_lo                    ;
         ld      rpm_diff_hi,rpm_2past_hi                   ; transfer the past into the difference
         ld      rpm_diff_lo,rpm_2past_lo                   ;
         sub     rpm_diff_lo,rpm_past_lo                    ; find the difference
         sbc     rpm_diff_hi,rpm_past_hi                    ;
         tm      rpm_diff_hi,#10000000b                     ; test for neg number
         jr      z,RPM_TIME_FOUND                           ; if the time is correct then jump
         ld      rpm_diff_hi,rpm_past_hi                    ; transfer the temp into the difference
         ld      rpm_diff_lo,rpm_past_lo                    ;
         sub     rpm_diff_lo,rpm_2past_lo                   ; find the difference
         sbc     rpm_diff_hi,rpm_2past_hi                   ;
RPM_TIME_FOUND:
         ld      rpm_period_hi,rpm_diff_hi                  ; transfer the difference to the period
         ld      rpm_period_lo,rpm_diff_lo                  ;

;-----------------------------------------------------------------------
; Found the period test for range
;-----------------------------------------------------------------------

         cp      rpm_period_hi,#12D                         ; test for a period of at least 6.144mS
         jp      ult,SKIPC                                  ; if the period is less then skip counting
         clr     UpDown                                     ; clear the direction timer


;-----------------------------------------------------------------------
; Position counter
;-----------------------------------------------------------------------

         cp      STATE,#1d                                  ; test the up direction state
```

```
        jr      z,DECPCOUNT                 ; if so then dec the counter
        cp      STATE,#2d                   ; test the up direction state
        jr      z,DECPCOUNT                 ; if so then dec the counter
        cp      STATE,#6d                   ; test the STOP state
        jr      z,DECPCOUNT                 ; if so then dec the counter

INCPCOUNT:
        inc     POSITION_LO                 ; increase the position counter low byte
        jr      nz,POSDONE                  ; if done return
        inc     POSITION_HI                 ; increase the position counter hi byte
        jr      POSDONE

DECPCOUNT:
        cp      POSITION_LO,#00             ; test for the roll number
        jr      z,DECPROLL                  ; if so the branch
        dec     POSITION_LO                 ; decrease the position counter low byte
        jr      POSDONE
DECPROLL:
        dec     POSITION_LO                 ; decrease the position counter low byte
        dec     POSITION_HI                 ; decrease the position counter hi byte
        jr      POSDONE

POSDONE:
;------------------------------------------------------------------------
; Enable the interrupts.
;------------------------------------------------------------------------
        ei
;------------------------------------------------------------------------
; Find the max force in the period
;------------------------------------------------------------------------

        cp      FORCE_IGNORE,#00            ; test for the force ignore active
        jr      nz,NOT_DELAY
        cp      rpm_period_hi,MAX_F_HI      ; test for a new max force
        jr      ult,NOT_MAX                 ; if not the max force then skip updating
        cp      rpm_period_lo,MAX_F_LO      ;
        jr      ult,NOT_MAX                 ;
SaveHigher:
        ld      MAX_F_HI,rpm_period_hi      ; transfer the max force data
        ld      MAX_F_LO,rpm_period_lo      ;
        cp      L_A_C,#4BH                  ; test for learn limit and force
        jr      ult,NOT_MAX                 ; if not then skip
        push    RP                          ; set the rp
        srp     #ForceTable2                ;
        ld      @forceaddress,MAX_F_HI      ; save the value into table
        inc     forceaddress                ;
        ld      @forceaddress,MAX_F_LO      ;
        dec     forceaddress                ;
        pop     RP
NOT_MAX:
        tm      POSITION_LO,#001111b        ; test for the 32th step
        jr      nz,NOT_DELAY                ;
                                            ; transfer to direction if L-A-C > 44
        ld      P32_MAX_HI,MAX_F_HI         ; transfer the value
        ld      P32_MAX_LO,MAX_F_LO
NOT_DELAY:
```

```
;-----------------------------------------------------------------------
; Force table entry
;-----------------------------------------------------------------------
        cp      L_A_C,#4CH                  ; test for the down direction
        jr      nz,N4C                      ; if not then skip around
        cp      POSITION_LO,#00             ; test for the position to increment
        jr      nz,N4E                      ; if not then skip
        clr     MAX_F_HI                    ; clear the max to get max
        clr     MAX_F_LO                    ; for the position window
        dec     ForceAddress                ; find the next address
        dec     ForceAddress
        cp      ForceAddress,#Force0Hi      ; test the range
        jr      uge,N4E                     ; if so skip
        ld      ForceAddress,#Force0Hi
N4C:
        cp      L_A_C,#4EH                  ; test for the up direction learn
        jr      nz,N4E                      ; if not then skip around
        cp      POSITION_LO,#0FFH           ; test for the position to increment
        jr      nz,N4E                      ; if not then skip
        clr     MAX_F_HI                    ; clear the max to get max
        clr     MAX_F_LO                    ; for the position window
        inc     ForceAddress                ; increment the pointer
        inc     ForceAddress                ; increment the pointer
        cp      ForceAddress,#Force14Hi     ; test for range
        jr      ule,N4E                     ; if in range skip
        ld      ForceAddress,#Force14Hi     ; else force address
N4E:
;-----------------------------------------------------------------------
; Look for the pass point
;-----------------------------------------------------------------------
        cp      AOBSSTATE,#00               ; test for aobs ok
        jr      z,AOBSRPMS                  ; if so skip the rpm count time out
        inc     AOBSRPM                     ; increment the timer counter
        cp      AOBSRPM,#MAXAR              ; test for too many
        jr      nz,AOBSRPMS                 ; if not skip
RPMOBS:
        ld      OBS_FLAG,#0CCH              ; else set the flag for aobs
AOBSRPMS:
        cp      AOBSSTATUS,#00              ; test for a obs blocked
        jr      nz,OBSBLOCK                 ; if the protector is blocked the jump
        inc     AOBSNB                      ; increase the aobs not blocked distance
        jr      AOBSDONE
OBSBLOCK:
        INC     AOBSB                       ; increase the aob blocked distance
AOBSDONE:
        cp      AOBSSTATE,#07               ; test for the max state
        jr      ule,STATEOK                 ; if in bounds then continue
        clr     AOBSSTATE
STATEOK:
        cp      AOBSSTATE,#00               ; test for the state number
        jr      z,state0
        cp      AOBSSTATE,#01               ; test for the state number
        jr      z,state1
        cp      AOBSSTATE,#02               ; test for the state number
```

```
        jr      z,state2
        cp      AOBSSTATE,#03                   ; test for the state number
        jr      z,state3
        cp      AOBSSTATE,#04                   ; test for the state number
        jr      z,state4
        cp      AOBSSTATE,#05                   ; test for the state number
        jr      z,state5
        cp      AOBSSTATE,#06                   ; test for the state number
        jr      z,state6
state7:
        cp      L_A_C,#4BH                      ; test for learn limits
        jr      ule,NoForceAddress
        ld      ForceAddress,#Force1Hi          ; set the force address
        cp      L_A_C,#4CH                      ; test for the down direction
        jr      nz,UpForceAdd
        ld      ForceAddress,#Force0Hi          ; set the force address
UpForceAdd:
        clr     MAX_F_HI                        ; clear the max force
        clr     MAX_F_LO
NoForceAddress:
        clr     AOBSRPM                         ; clear all rpm counts during

        cp      L_A_C,#42H                      ; test for learn mode
        jr      uge,SkipFlagTest                ; if so winflag is useless

        cp      WIN_FLAG,#00                    ; test for the first cycle
        jr      z,ClearPassPoint

SkipFlagTest:
        cp      STATE,#04d                      ; test for traveling down
        jr      nz,SkipPassPoint                ; if not the skip the pass point clear

ClearPassPoint:
        di

        clr     POSITION_LO                     ; clear the position reg
        clr     POSITION_HI                     ;

        ei

SkipPassPoint:
        ld      AOBS_FLAG,#01d                  ; set the flag for got pass point
        jr      ASDONE
state4:
        cp      AOBSB,#00                       ; test for not blocked
        jr      TN1


state3:
        cp      AOBSNB,#MINAR                   ; test for the min blockage
        jr      TN2

state6:
state2:
        cp      AOBSNB,#00                      ; test for not blocked
TN1:
```

```
        jr      z,STATEDONE             ; if still waiting loop
        inc     AOBSSTATE               ; set the next state
        jr      STATEDONE

state5:
state1:
        cp      AOBSB,#MINAR            ; test for the min blockage
TN2:
        jr      ult,STATEDONE           ; if not try again
ASDONE:
        inc     AOBSSTATE               ; set the next state
        clr     AOBSNB                  ; clear the not blocked
        clr     AOBSB                   ; clear the blocked
        jr      STATEDONE               ;

state0:
        cp      AOBSB,#00               ; test for the first blockage
        jr      z,STATEDONE             ; if no block skip
        push    rp                      ; save the rp
        srp     #FORCE_GRP              ; set the new value
        cp      L_A_C,#47h              ; test for the state for storage
        jr      nz,NOSTORE              ; if not then continue
        clr     UP_LIM_HI               ;
        clr     UP_LIM_LO
        sub     UP_LIM_LO,position_lo   ; get the - of the count
        sbc     UP_LIM_HI,position_hi   ;
        call    FIND_WINDOW             ; find the window

NOSTORE:
        di
        push    position_lo             ; save the lo position
        cp      WIN_FLAG,#00            ; test for the window being active
        jr      z,WIN_SKIP              ; if inactive skip
        cp      position_hi,#00         ; test for pos or neg
        jr      z,WINTEST               ; jump if the value POS < 256
negwin:
        cp      position_hi,#0FFH       ; test for < 256
        jr      nz,WINERROR             ; if not then a error
        com     position_lo             ; neg the value
WINTEST:
        cp      position_lo,PWINDOW     ; compare the pos value of window
        jr      ULE,WIN_SKIP            ; if within then ok

WINERROR:
        ld      OBS_FLAG,#0CCH          ; set the flag for aobs
        pop     position_lo             ; reset the position
        pop     rp                      ; reset the rp
        jr      STATEDONE               ; done

WIN_SKIP:
        pop     position_lo             ; reset the position
        pop     rp                      ; reset the rp
        inc     AOBSSTATE               ; set the next state
STATEDONE:
```

```
;---------------------------------------------------------------------
; Look for the pass point   end
;---------------------------------------------------------------------

TULS:
INCRPM:
        di
        inc     RPM_COUNT                       ; increase the rpm count
        inc     RPM_ACOUNT                      ; increase the rpm count
        ei
SKIPC:
        di
        ld      rpm_time_out,#15D               ; set the rpm max period as 30mS
        ld      BRPM_TIME_OUT,#15D              ; set the rpm max period as 30mS
                                                ; if rpm not updated by then reverse

        ei
SKIPPEDGE:
        pop     rp                              ; return the rp

        iret                                    ; return



;---------------------------------------------------------------------
;       Find the window size from the up limit setting
;---------------------------------------------------------------------
FIND_WINDOW:
        cp      UP_LIM_HI,#0FAh                 ; test for the shortest distance
        jr      UGT,S100D                       ; if so set window to 100D
        cp      UP_LIM_HI,#0F8h                 ; test for the mid distance
        jr      UGT,S150D                       ; if so then set the window to 150D
        ld      PWINDOW,#200D                   ; set the window to 200D
        ret
S150D:
        ld      PWINDOW,#150D                   ; set the window to 150D
        ret
S100D:
        ld      PWINDOW,#100D                   ; set the window to 100D
        ret


;---------------------------------------------------------------------
;
;       Read the force according to the position
;
;---------------------------------------------------------------------
ReadForce:
        push    RP                              ; set the RP
        srp     #ForceTable2                    ;
        ld      forcetemp,POSITION_HI           ; get the present position of the operator
        com     forcetemp                       ; invert the number
        cp      forcetemp,#10H                  ; test for the set to address 0 values
        jr      uge,SetAddress00                ;
        inc     forcetemp                       ; add 1 for address
        cp      forcetemp,#0DH                  ; test for in range
```

```
        jr      uge,SetAddressD                         ; if not set the top address

SetForce:

        rcf                                             ; *2
        rlc     forcetemp                               ;
        add     forcetemp,#Force0Hi                     ; add the start address
        push    forcetemp                               ; save value
        di
        ld      UP_FORCE_HI,@forcetemp                  ; read the value
        inc     forcetemp                               ; save address
        ld      UP_FORCE_LO,@forcetemp                  ;
        add     UP_FORCE_LO,ForceAddLo                  ; add adder
        adc     UP_FORCE_HI,ForceAddHi                  ;
        pop     forcetemp                               ; reset address
        ei
        di
        ld      DN_FORCE_HI,@forcetemp                  ; read the value
        inc     forcetemp
        ld      DN_FORCE_LO,@forcetemp                  ;
        add     DN_FORCE_LO,ForceAddLo                  ; add adder
        adc     DN_FORCE_HI,ForceAddHi
        ei
        pop     RP                                      ; then return
SkipForceRead:
        ret

SetAddress00:
        clr     forcetemp                               ; set the address
        jr      SetForce

SetAddressD:
        ld      forcetemp,#0DH                          ; set the address
        jr      SetForce

;-----------------------------------------------------------------------
;
;       Read the Limits
;
;-----------------------------------------------------------------------

ReadLimits:
        push    rp                                      ; set the RP to  LEARNEE_GRP
        srp     #LEARNEE_GRP                            ;
        ld      SKIPRADIO,#0FFH                         ; turn off the radio
        ld      address,#AddressDownLimit               ; set non vol address to the down limit
        call    READMEMORY                              ; read the value
        di
        ld      DN_LIM_HI,mtemph                        ; recall from  nonvolital
        ld      DN_LIM_LO,mtempl                        ;
        ei

        ld      address,#AddressUpLimit                 ; set non vol address to the up limit
        call    READMEMORY                              ; read the values stored in memory
        di
        ld      UP_LIM_HI,mtemph                        ; update from nonvolital
```

```
        ld      UP_LIM_LO,mtempl                        ;
        ei
        clr     SKIPRADIO                               ; turn on the radio
        pop     rp                                      ; reset the RP
        ret


;***********************************************************************
;
; Timer 2 Interrupt used either for RS232 or Wall control
; Rs232 is set to 416uS    Wall control is set to 300uS
; Wall control state machine
; Status
;       0 =     If not low set gotswitch
;               Switch from discharge to charge P3 = 1001 XXXX
;               Test for hi after 4uS  switch = open
;               Test for hi after 30uS switch = light
;       1 =     Test for hi after 300uS switch = learn
;      10 =     Test for hi after 3mS  switch = vacation
;               Else switch = cmd
;      11 =     Switch state to discharge P3 = 1111 XXXX
;      15 =     Switch state to neg charge if led is to be lit
;                               P3 = 0110 XXXX
;               Else
;               Switch state to no charge P3 = 0000 XXXX
;      26 =     Switch state to discharge
;      29 =     Set Status to 0
;***********************************************************************
Timer2Int:


;       tm      P2,#01000000B                           ; test the RS232 only switch
;       jr      z.SkipLockRS232
;       jr      TestRs232                               ; if switch then just RS232
;
;SkipLockRS232:
        cp      RsMode,#0232d                           ; test for rs232 mode set
        jr      z,TestRs232                             ; if set do
        cp      RsTimer,#0FFH                           ; test the mode for RS232 Vs switches
        jr      z,TestSwitches                          ; if FF then test the switches
TestRs232:
        cp      T1Mirror,#RsPeriod                      ; test the period
        jp      nz,SetRsPeriod                          ; if set wrong then reset
        call    RS232                                   ; call the routine
        iret                                            ; return


TestSwitches:
        cp      STATUS,#0FFH                            ; test for the start position
        jp      nz,SkipVacFlashing                      ; if not skip testing vacation flashing

        cp      VACFLAG,#00H                            ; test for out of vacation
```

```
        jp      z,SkipVacFlashing               ; if out don't blink

        tm      VACFLASH,#10000000B             ; test for the 128mS
        jp      z,SkipVacFlashing               ; if out don't blink

        ld      STATUS,#30D                     ; set for the blink

SkipVacFlashing:
        inc     STATUS                          ; set to the next period
        cp      T1Mirror,#SwPeriod              ; test the period
        jp      nz,SetSwPeriod                  ; if set wrong then reset
        cp      STATUS,#0d                      ; State jump table
        jp      z,STATUS0                       ;
        cp      STATUS,#1d                      ;
        jp      z,STATUS1                       ;
        cp      STATUS,#10d                     ;
        jp      z,STATUS10                      ;
        cp      STATUS,#11d                     ;
        jp      z,STATUS11                      ;
        cp      STATUS,#15d                     ;
        jp      z,STATUS15                      ;
        cp      STATUS,#26d                     ;
        jp      z,STATUS26                      ;
        cp      STATUS,#29d                     ;
        jp      uge,STATUS29                    ;
StatusRet:
        iret
STATUS0:
        tm      P0,#11000000B                   ; test for both inputs low
        jr      z,SkipSettingGotSw1             ; if low skip seting
        inc     GotSwitch                       ; turn off the switches
SkipSettingGotSw1:                              ; use hist to test resistors
        ld      P01M,#00000100B                 ; set mode p00-p03 out p04-p07out
        or      P0,#1100000B                    ; turn both pins hi
        ld      P01M,#P01M_INIT                 ; set mode p00-p03 out p04-p07in
        nop                                     ; delay
        nop
        nop
        nop
        tm      P0,#11000000B                   ; test for both inputs low
        jr      z,SkipSettingGotSw2             ; if low skip seting
        inc     GotSwitch                       ; turn off the switches
SkipSettingGotSw2:                              ; use hist to test resistors

        push    TEMP                            ;
        ld      TEMP,P3
        and     TEMP,#00001111B                 ; turn both off
        or      TEMP,#10010000B                 ; turn on charge
        ld      P3,TEMP                         ;
        pop     TEMP
        nop                                     ; delay
        tm      P0,#10000000B                   ; test 4 uS later
        jr      nz,GotOpen                      ; if so then open
        nop
        nop
        nop
```

```
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        tm      P0,#10000000B           ; test 30uS out
        jp      nz,GotLight             ; if so then light
        iret

STATUS1:
        tm      P0,#10000000B           ; test 300uS later
        jp      nz,GotLearn             ; if so then got the learn
        iret

STATUS10:
        tm      P0,#10000000B           ; test 3mS later
        jp      nz,GotVac               ; if so then got the vac
        jp      GotCmd

STATUS11:
        or      P3,#11110000B           ; turn all on discharge
        iret

STATUS15:
        and     P3,#00001111B           ; turn off both outputs
        tcm     LearnLed,#00111111b     ; test for off
        jp      z,StatusRet             ; if so then return
        tm      LearnLed,#11000000B     ; test for radio blink mode
        jr      nz,SkipLedInc           ; if not skip inc timer
        inc     LearnLed                ;
SkipLedInc:
        or      P3,#01100000B           ; turn on the led
        iret
STATUS26:
        or      P3,#11110000B           ; set the discharge state
        iret

STATUS29:
        cp      STATUS,#30D             ; test for the blink
        jr      uge,BlinkTime
Status29:
        clr     GotSwitch               ; clear got a switch flag
        ld      STATUS,#0FFH            ; reset the machine
        iret                            ; return
BlinkTime:
        cp      STATUS,#60D             ; test for the end of the run
        jr      uge,Status29            ; if so return
```

```
        cp      STATUS,#45D             ; test for the led period
        jr      ult,STATUS11            ; if not then discahrge
        cp      STATUS,#56D             ;
        jr      uge,STATUS11            ;
        jr      STATUS15                ; else set the program led


SetSwPeriod:
        ld      T1Mirror,#SwPeriod      ; set the period
        jr      SetT1Period
SetRsPeriod:
        ld      T1Mirror,#RsPeriod      ; set the period
SetT1Period:
        ld      T1,T1Mirror             ;
        ld      TMR,#00001110B          ; turn on the timer
        iret                            ; retum one shoted

GotOpen:                                ; open decrement all
        call    DecrementCmd            ;
        call    DecrementLight          ;
        call    DecrementLearn          ;
        call    DecrementVacation       ;
        iret
GotLight:                               ; light
        cp      GotSwitch,#00           ; test for got switch
        jr      z,DoLight               ; if not then do the light
        iret                            ; else return
DoLight:
        call    DecrementCmd            ;
        call    IncrementLight          ;
        call    DecrementLeam           ;
        call    DecrementVacation       ;
        iret
GotLearn:
        cp      GotSwitch,#00           ; test for got switch
        jr      z,DoLeam                ; if not then do the learn
        iret                            ; else return
DoLearn:
        call    DecrementCmd            ;
        call    DecrementLight          ;
        call    IncrementLearn          ;
        call    DecrementVacation       ;
        iret
GotVac:
        cp      GotSwitch,#00           ; test for got switch
        jr      z,DoVac                 ; if not then do the Vac
        iret                            ; else return
DoVac:
        call    DecrementCmd            ;
        call    DecrementLight          ;
        call    DecrementLearn          ;
        call    IncrementVacation       ;
        iret
GotCmd:
        cp      GotSwitch,#00           ; test for got switch
        jr      z,DoCmd                 ; if not then do the cmd
```

```
        iret                                    ; else return
DoCmd:
        call    IncrementCmd                    ;
        call    DecrementLight                  ;
        call    DecrementLearn                  ;
        call    DecrementVacation               ;
        iret

IncrementCmd:
        inc     GotSwitch                       ; set the got a switch flag
        cp      CMD_DEB,#0FFH                   ; test for at the top
        jr      z,SkipCmdInc                    ; if so then skip
        inc     CMD_DEB                         ; inc
        inc     BCMD_DEB
        cp      CMD_DEB,#9d                     ; test for cmd
        jr      nz,SkipCmdInc                   ; if not the skip Cmd

        ld      CMD_DEB,#0FFH                   ; set deb back to top
        ld      BCMD_DEB,CMD_DEB
CmdSet:
        cp      L_A_C,#42H                      ; test for learn seq
        jr      ult,NotInLearn                  ; if not in learn skip
        ld      L_A_C,#042h                     ; set the next level of force
        jr      SkipCmdInc                      ; skip command
NotInLearn:
        cp      LEARNT,#0FFH                    ; test for learn mode
        jr      z,NLearnACmd                    ; if not
        ld      L_A_C,#042h                     ; set the next level
        ld      FORCES,#03                      ; set the starting force to lowest
        ld      LearnLed,#00111111b             ; turn off the led
        ld      LEARNT,#0FFH                    ; set the learn timer
        ld      LEARNDB,#0FFH                   ; set the learn debounce
        jr      SkipCmdInc                      ; DO NOT issue a command
NLearnACmd:
        ld      LAST_CMD,#055H                  ; set the last command as wall cmd
        ld      SW_DATA,#CMD_SW                 ; set the switch data as command
SkipCmdInc:
        ret

DecrementCmd:
        inc     GotSwitch                       ; set the got a switch flag
        cp      CMD_DEB,#00                     ; test for the bottom
        jr      z,SkipCmdDec                    ; if so then skip
        dec     CMD_DEB                         ; dec
        dec     BCMD_DEB
        cp      CMD_DEB,#0F6H                   ; test for release
        jr      nz,SkipCmdDec                   ; if not done
        clr     CMD_DEB                         ;
        clr     BCMD_DEB
SkipCmdDec:
        ret

IncrementLight:
        cp      LIGHT_DEB,#0FFH                 ; test for at the top
        jr      z,SkipLightInc                  ; if so then skip
```

```
        inc     LIGHT_DEB                               ; inc
        cp      LIGHT_DEB,#9d                           ; test for light
        jr      nz,SkipLightInc                         ; if not skip light cmd
LightSet:
        cp      LEARNT,#0FFH                            ; test for learn mode
        jr      z,NotInLearnLight
        cp      STATE,#2d                               ; test for up position
        jr      nz,NotInLearnLight
JogUp:
        ld      Jog,#055H                               ; set the jog
        jr      SkipLightInc
NotInLearnLight:
        ld      LIGHT_DEB,#0FFH                         ; set deb to top
        ld      SW_DATA,#LIGHT_SW                       ; set the switch data

SkipLightInc:
        ret

DecrementLight:
        cp      LIGHT_DEB,#00                           ; test for the bottom
        jr      z,SkipLightDec                          ; if so then skip
        dec     LIGHT_DEB                               ; dec
        cp      LIGHT_DEB,#0F6H                         ; test for release
        jr      nz,SkipLightDec                         ; if not deon
        clr     LIGHT_DEB
SkipLightDec:
        ret

IncrementVacation:
        cp      VAC_DEB,#0FFH                           ; test for at the top
        jr      z,SkipVacInc                            ; if so then skip
        inc     VAC_DEB                                 ; inc
        cp      VAC_DEB,#55d                            ; test for vacation activation
        jr      nz,SkipVacInc                           ; if not exit
VacSet:
        cp      LEARNT,#0FFH                            ; test for learn mode
        jr      z,NotInLearnVac
        cp      STATE,#2d                               ; test for up position
        jr      nz,NotInLearnVac
JogDown:
        ld      Jog,#0AAH                               ; jog down
        jr      SkipVacInc

NotInLearnVac:
        ld      VAC_DEB,#0FFH                           ; set deb
        ld      VACCHANGE,#0AAH                         ; set the toggle data

SkipVacInc:
        ret

DecrementVacation:
        cp      VAC_DEB,#00                             ; test for the bottom
        jr      z,SkipVacDec                            ; if so then skip
        dec     VAC_DEB                                 ; dec
        cp      VAC_DEB,#(0FFH-55D)                     ; test for reset level
        jr      nz,SkipVacDec                           ; if not then return
```

```
        clr     VAC_DEB                         ; reset the debouncer
SkipVacDec:
        ret

IncrementLearn:
        cp      STATE,#AUTO_REV                 ; test for motion states
        jr      z,SkipLearnInc                  ; if so then do not inc
        cp      STATE,#UP_DIRECTION             ;
        jr      z,SkipLearnInc                  ;
        cp      STATE,#DN_DIRECTION             ;
        jr      z,SkipLearnInc                  ;
        cp      LEARNDB,#0FFH                   ; test for at the top
        jr      z,SkipLearnInc                  ; if so then skip
        inc     LEARNDB                         ; inc
        cp      LEARNDB,#9D                     ; test for learn activation
        jr      nz,SkipLearnInc                 ; if not then exit
LearnSet:
        ld      LEARNDB,#0FFH                   ; set deb
        clr     LEARNT                          ; clear the learn timer
        ld      LearnLed,#10000000B             ; turn on the learn led
        cp      VACFLAG,#00H                    ; test the flag for out of vacation
        jr      z,SkipVacChange
        ld      VACCHANGE,#0AAH                 ; if in vacation change it
SkipVacChange:
SkipLearnInc:
        ret

DecrementLeam
        cp      LEARNDB,#00                     ; test for the bottom
        jr      z,SkipLearnDec                  ; if so then skip
        dec     LEARNDB                         ; dec
        cp      LEARNDB,#0F6H                   ; test for reset level
        jr      nz,SkipVacDec                   ; if not then return
        clr     LEARNDB                         ; reset the debouncer
SkipLearnDec:
        ret

;****************************************************************
;
; Temperature measurement
;****************************************************************
;
TempMeasure:
        .IF     E21
        xor     P1,#00000001B                   ; Kick the external dog
        .ELSE
        WDT                                     ; KICK THE DOG
        .ENDIF
        di
        ld      ForceAddHi,#0FFH                ; clear the value
        ld      ForceAddLo,#0FFH                ;
        ld      TMR,#00001011B                  ; load the timer
        or      P2,#00000001b                   ; turn on the temperature rc
        ld      TMR,#00001010B                  ; run
LoopTillTemp1:
        tm      P2,#00100000B                   ; test for done
        jr      nz,TempMeasured
```

```
        cp      T0,#010H                ; test for lower roll
        jr      ugt,LoopTillTemp1
        .IF     E21
        xor     P1,#00000001B           ; Kick the external dog
        .ELSE
        WDT                             ; KICK THE DOG
        .ENDIF
LoopTillTemp2:
        tm      P2,#00100000B           ; test for done
        jr      nz,TempMeasured
        cp      T0,#0EEH                ; test for lower roll
        jr      ult,LoopTillTemp2
Roll:
        dec     ForceAddHi              ;
        cp      ForceAddHi,#0EFH        ; should be two  test for too long
        jp      ule,ErrorSetMaxTemp     ; if so set error
        jr      LoopTillTemp1           ; loop till done


TempMeasured:
        ld      ForceAddLo,T0           ; set the value
        com     ForceAddHi
        com     ForceAddLo

                                        ; house cleaning

        ld      AOBSTEST,#11D           ; reset the test timer
        or      AOBSF,#00000010B        ; set the flag for got a aobs
        clr     AOBSSTATUS              ; clear the aobs set state

        .IF     E21
        xor     P1,#00000001B           ; Kick the external dog
        .ELSE
        WDT                             ; KICK THE DOG
        .ENDIF
        .IF RTD
TempOk:
        cp      ForceAddHi,#00d         ; test for count < 100H
        jr      z,Msb00                 ;
        cp      ForceAddHi,#01d         ; test for count < 200H
        jr      z,Msb10                 ;
        cp      ForceAddHi,#11d         ; test for < 1100h
        jr      ult,Tn15                ;
        cp      ForceAddHi,#14h         ; test for < 1400H
        jr      ult,Tn40                ;
        jp      ErrorSetMaxTemp         ; else error

Msb00:
        cp      ForceAddLo,#07h         ; test for the bounds
        jr      ule,ErrorSetMaxTemp     ; if so then error
        cp      ForceAddLo,#2Ah         ; test for 85 deg
        jr      ult,T85                 ; if so then jump
        cp      ForceAddLo,#6Fh         ; test for 60 deg
        jr      ult,T60                 ; if so then jump
        jr      T35                     ; else it is 35 deg
```

```
Msb10:
        cp      ForceAddLo,#4Eh         ; test for 35 deg
        jr      ult,T35                 ; if so then jump
        jr      T10                     ; else it is 10 deg

T85:
        ld      Temperature,#125D       ; set the temperature
        ld      ForceAddHi,#000         ; set the force
        ld      ForceAddLo,#0FAH        ;
        jr      ExitTemperature         ; test motor for too cold and exit

T60:
        ld      Temperature,#100D       ; set the temperature
        ld      ForceAddHi,#001H        ; set the force
        ld      ForceAddLo,#00EH        ;
        jr      ExitTemperature         ; test motor for too cold and exit

T35:
        ld      Temperature,#75D        ; set the temperature
        ld      ForceAddHi,#001H        ; set the force
        ld      ForceAddLo,#022H        ;
        jr      ExitTemperature         ; test motor for too cold and exit

T10:
        ld      Temperature,#50D        ; set the temperature
        ld      ForceAddHi,#001H        ; set the force
        ld      ForceAddLo,#040H        ;
        jr      ExitTemperature         ; test motor for too cold and exit

Tn15:
        ld      Temperature,#25D        ; set the temperature
        ld      ForceAddHi,#001H        ; set the force
        ld      ForceAddLo,#05EH        ;
        jr      ExitTemperature         ; test motor for too cold and exit

Tn40:
        ld      Temperature,#0D         ; set the temperature
        ld      ForceAddHi,#001H        ; set the force
        ld      ForceAddLo,#090H        ;
        jr      ExitTemperature         ; test motc- for too cold and exit
        .ELSE
TempOk:
        cp      ForceAddHi,#00d         ; test for the first 512uS
        jr      z,LessThen512           ;
        cp      ForceAddHi,#01d         ; test for the 1024 limit
        jr      z,LessThen1024          ;

        jp      ErrorSetMaxTemp         ; else set to max


LessThen512:
        cp      ForceAddLo,#0D0H        ; test for too low
        jr      ule,ErrorSetMaxTemp     ; if so set error values
        cp      ForceAddLo,#0EEH        ; test for 85C
        jr      ult,T85C                ; if so set the temp
```

```
        jr      T60C

LessThen1024:
        cp      ForceAddLo,#0BH          ; test for 60 C
        jr      ult,T60C                 ; if so set
        cp      ForceAddLo,#26H          ; test for 35C
        jr      ult,T35C                 ; if so set the temp
        cp      ForceAddLo,#43H          ; test for 10C
        jr      ult,T10C                 ; if so set the temp
        cp      ForceAddLo,#60H          ; test for -15C
        jr      ult,TN15C                ; if so then set the temp
        cp      ForceAddLo,#80H          ; test for -40C
        jr      ult,TN40C                ; if so then set the temp
        jr      ErrorSetMaxTemp

T85C:
        ld      Temperature,#125D        ; set the temperature
        jr      ExitTemperature          ; test motor for too cold and exit

T60C:
        ld      Temperature,#100D        ; set the temperature
        jr      ExitTemperature          ; test motor for too cold and exit

T35C:
        ld      Temperature,#75D         ; set the temperature
        jr      ExitTemperature          ; test motor for too cold and exit

T10C:
        ld      Temperature,#50D         ; set the temperature
        jr      ExitTemperature          ; test motor for too cold and exit

TN15C:
        ld      Temperature,#25D         ; set the temperature
        jr      ExitTemperature          ; test motor for too cold and exit

TN40C:
        ld      Temperature,#0D          ; set the temperature
        jr      ExitTemperature          ; test motor for too cold and exit

        .ENDIF



ErrorSetMaxTemp:
        .IF     E21
        xor     P1,#00000001B            ; Kick the external dog
        .ELSE
        WDT                              ; KICK THE DOG
        .ENDIF
        ld      ForceAddHi,#00h          ; set the force to .5mS
        ld      ForceAddLo,#0FFH         ;
        ld      Temperature,#85d+40D     ; set the temperature to the max
ExitTemperature:
        cp      MotorTempHi,Temperature  ; test for the motor value too low
```

```
        jr      uge,MotorTempDone                  ; if hoter or = don't change
        ld      MotorTempHi,Temperature            ; else set =
MotorTempDone:
        and     P2,#11111110b                      ; turn off the temperature rc

        .IF     ForceTempCompFlag
        .ELSE
        ld      ForceAddHi,#00h                    ; set the force to .5mS
        ld      ForceAddLo,#0FFH                   ;
        .ENDIF

        .IF     TempMeasureFlag
        .ELSE
        ld      Temperature,#85d+40D               ; set the temperature to the max
        .ENDIF

        ei                                         ; reenable the interrupts
        ret

.end
```

14035052702